1. Documentation . . . 10
2. WordWriter for Windows Version 4 . . . 10
2.1 Introduction . . . 10
2.1.1 What is WordWriter? . . . 10
2.1.2 Features and Benefits . . . 11
2.1.3 WordWriter Editions . . . 11
2.1.4 New Features . . . 12
2.1.5 Requirements . . . 13
2.1.6 Frequently Asked Questions . . . 14
2.1.7 Office 2007 Support . . . 15
2.1.8 Change Log . . . 15
2.2 Quickstart . . . 17
2.2.1 Create a Document with WordTemplate . . . 17
2.2.2 Create a Document with WordApplication . . . 19
2.2.3 Upgrading from WordWriter Version 3 to WordWriter Version 4 . . . 21
2.3 WordTemplate in Depth . . . 22
2.3.1 Creating a Template . . . 22
2.3.2 Creating Merge Fields . . . 27
2.3.3 Using an Array as a Data Source . . . 30
2.3.4 Using a Database as a Data Source . . . 32
2.3.5 Importing from Multiple Data Sources . . . 34
2.3.6 Importing Multiple Rows . . . 35
2.3.7 Creating a Grouped and Nested Document . . . 37
2.3.8 WordTemplate Output Options . . . 44
2.3.9 Adding Formulas to a Document . . . 47
2.3.10 Inserting an Image . . . 50
2.3.11 Formatting Merge Fields . . . 51
2.4 WordApplication in Depth . . . 56
2.4.1 Creating a Paragraph . . . 56
2.4.2 Formatting Text . . . 58
2.4.3 Creating Lists . . . 60
2.4.4 Creating Tables . . . 62
2.4.5 Creating Sections . . . 63
2.4.6 Adding Images . . . 64
2.4.7 Inserting Watermarks . . . 65
2.4.8 Importing Data . . . 66
2.4.9 WordApplication Output Options . . . 68
2.5 SharePoint Integration in Depth . . . 69
2.5.1 Exporting list as Word document . . . 69
2.5.2 SharePoint Solution - Word Export Plus . . . 79
2.6 WordWriter in ASP . . . 82
2.6.1 How to Use WordWriter in ASP . . . 82
2.6.2 ASP Reference . . . 84
2.6.2.1 WordTemplate Object . . . 84
2.6.2.2 BookmarkFieldMarkers Method . . . 84
2.6.2.3 Bookmarks Property . . . 85
2.6.2.4 ContentType Property . . . 86
2.6.2.5 CultureInfo Property . . . 86
2.6.2.6 FieldMarkers Property . . . 87
2.6.2.7 LicenseKey Property . . . 87
2.6.2.8 Open Method . . . 87
2.6.2.9 Process Method . . . 88
2.6.2.10 Save Method . . . 89
2.6.2.11 SetDataSource Method . . . 90
2.6.2.12 SetRepeatBlock Method . . . 91
2.6.2.13 Version Property . . . 92
2.7 Programmer's Reference . . . 93
2.7.1 WordApplication Object Hierarchy Diagram . . . 94
2.7.2 WordApplication Index . . . 94
2.7.3 WordTemplate Programmer's Reference . . . 103
2.7.3.1 WordTemplate . . . 103
2.7.3.1.1 WordTemplate.ALL_ROWS . . . 108
2.7.3.1.2 WordTemplate.BookmarkFieldMarkers(String) . . . 108
2.7.3.1.3 WordTemplate.Bookmarks . . . 109
2.7.3.1.4 WordTemplate.ContentType . . . 110
2.7.3.1.5 WordTemplate.CultureInfo . . . 111
2.7.3.1.6 WordTemplate.DataSourceSeparator . . . 112
2.7.3.1.7 WordTemplate.EnableNEXTFields . . . 113
2.7.3.1.8 WordTemplate.FieldMarkers . . . 113
2.7.3.1.9 WordTemplate.LicenseKey . . . 114
2.7.3.1.10 WordTemplate.Open . . . 115
2.7.3.1.11 WordTemplate.PreserveFields . . . 117
2.7.3.1.12 WordTemplate.Process() . . . 118
2.7.3.1.13 WordTemplate.Save . . . 119
2.7.3.1.14 WordTemplate.SaveExtension . . . 123

2.7.3.1.15 WordTemplate.SetDataSource . . . 124
2.7.3.1.16 WordTemplate.SetMailMerge . . . 143
2.7.3.1.17 WordTemplate.SetRepeatBlock . . . 164
2.7.3.1.18 WordTemplate.UpdateIfFields . . . 186
2.7.3.1.19 WordTemplate.Version . . . 186
2.7.3.1.20 WordTemplate.WordTemplate() . . . 187
2.7.4 WordApplication Programmer's Reference . . . 188
2.7.4.1 AbsolutePositioning . . . 189
2.7.4.1.1 AbsolutePositioning.Height . . . 190
2.7.4.1.2 AbsolutePositioning.HeightExact . . . 190
2.7.4.1.3 AbsolutePositioning.HorizontalOrigin . . . 191
2.7.4.1.4 AbsolutePositioning.OriginX . . . 191
2.7.4.1.5 AbsolutePositioning.OriginY . . . 192
2.7.4.1.6 AbsolutePositioning.VerticalOrigin . . . 193
2.7.4.1.7 AbsolutePositioning.Width . . . 193
2.7.4.1.8 AbsolutePositioning.X . . . 194
2.7.4.1.9 AbsolutePositioning.Y . . . 194
2.7.4.2 AutoFormatter . . . 195
2.7.4.2.1 AutoFormatter.Edge . . . 195
2.7.4.2.2 AutoFormatter.FormatCell(TableCell, Int32, Int32, AutoFormatter.Edge) . . . 196
2.7.4.3 Bookmark . . . 196
2.7.4.3.1 Bookmark.Name . . . 197
2.7.4.4 Border . . . 198
2.7.4.4.1 Border.Color . . . 199
2.7.4.4.2 Border.HasShadow . . . 200
2.7.4.4.3 Border.LineStyle . . . 200
2.7.4.4.4 Border.LineWidth . . . 201
2.7.4.4.5 Border.Location . . . 202
2.7.4.4.6 Border.Padding . . . 203
2.7.4.4.7 Border.Reverse . . . 203
2.7.4.4.8 Border.Style . . . 204
2.7.4.5 CharacterRun . . . 205
2.7.4.5.1 CharacterRun.Deleted . . . 206
2.7.4.5.2 CharacterRun.DeletedBy . . . 207
2.7.4.5.3 CharacterRun.DeletedTime . . . 207
2.7.4.5.4 CharacterRun.Font . . . 208
2.7.4.5.5 CharacterRun.ModifiedBy . . . 208
2.7.4.5.6 CharacterRun.ModifiedTime . . . 209
2.7.4.5.7 CharacterRun.New . . . 209
2.7.4.5.8 CharacterRun.PropertiesChangedBy . . . 210
2.7.4.5.9 CharacterRun.PropertiesChangedTime . . . 210
2.7.4.5.10 CharacterRun.Style . . . 210
2.7.4.6 CompatibilitySettings . . . 211
2.7.4.6.1 CompatibilitySettings.AdjustTableLineHeight . . . 213
2.7.4.6.2 CompatibilitySettings.AlignRowsIndependent . . . 214
2.7.4.6.3 CompatibilitySettings.AllowLeading . . . 214
2.7.4.6.4 CompatibilitySettings.AllowSnapInCells . . . 215
2.7.4.6.5 CompatibilitySettings.AllowTableBreaks . . . 216
2.7.4.6.6 CompatibilitySettings.ApplyBreakingRules . . . 216
2.7.4.6.7 CompatibilitySettings.AutospaceLike95 . . . 217
2.7.4.6.8 CompatibilitySettings.BalanceColumns . . . 217
2.7.4.6.9 CompatibilitySettings.BorderRulesLike6 . . . 218
2.7.4.6.10 CompatibilitySettings.ConvertMailMergeEsc . . . 219
2.7.4.6.11 CompatibilitySettings.ExactOnTop . . . 219
2.7.4.6.12 CompatibilitySettings.ExpandShiftReturn . . . 220
2.7.4.6.13 CompatibilitySettings.FootnotesLike97 . . . 221
2.7.4.6.14 CompatibilitySettings.GrowTableAutoFit . . . 221
2.7.4.6.15 CompatibilitySettings.JustifyLikeWP6 . . . 222
2.7.4.6.16 CompatibilitySettings.LeaveBackslashAlone . . . 223
2.7.4.6.17 CompatibilitySettings.LineBreaksLike97 . . . 223
2.7.4.6.18 CompatibilitySettings.LineWrapLike6 . . . 224
2.7.4.6.19 CompatibilitySettings.MapPrintColor . . . 224
2.7.4.6.20 CompatibilitySettings.MirrorMargins . . . 225
2.7.4.6.21 CompatibilitySettings.NoSpaceForRaisedLower . . . 226
2.7.4.6.22 CompatibilitySettings.NoTabForIndent . . . 226
2.7.4.6.23 CompatibilitySettings.ShapesLike97 . . . 227
2.7.4.6.24 CompatibilitySettings.ShowBreaksInFrames . . . 228
2.7.4.6.25 CompatibilitySettings.SnapPageBorder . . . 228
2.7.4.6.26 CompatibilitySettings.SpaceLikeWP5 . . . 229
2.7.4.6.27 CompatibilitySettings.SubBasedOnSize . . . 229
2.7.4.6.28 CompatibilitySettings.SuppressExtraAfter . . . 230
2.7.4.6.29 CompatibilitySettings.SuppressSpaceAfterPageBreak . . . 231
2.7.4.6.30 CompatibilitySettings.SuppressSpacingLike5 . . . 231
2.7.4.6.31 CompatibilitySettings.SuppressSpacingLikeWP . . . 232
2.7.4.6.32 CompatibilitySettings.SupressTopSpacing . . . 233

2.7.4.6.33 CompatibilitySettings.SwapBorders . . . 233
2.7.4.6.34 CompatibilitySettings.TableRowsApart . . . 234
2.7.4.6.35 CompatibilitySettings.TruncateFontHeight . . . 234
2.7.4.6.36 CompatibilitySettings.UnderlineTrailingSpaces . . . 235
2.7.4.6.37 CompatibilitySettings.UseAsianBreakingRules . . . 236
2.7.4.6.38 CompatibilitySettings.UseLargerSmallCaps . . . 236
2.7.4.6.39 CompatibilitySettings.UsePrinterMetrics . . . 237
2.7.4.6.40 CompatibilitySettings.UseTableRawWidth . . . 238
2.7.4.6.41 CompatibilitySettings.UseTableStylesLike2002 . . . 238
2.7.4.6.42 CompatibilitySettings.WrapTextWithPunct . . . 239
2.7.4.6.43 CompatibilitySettings.WrapTrailingSpaces . . . 240
2.7.4.7 DataImportProperties . . . 240
2.7.4.7.1 DataImportProperties.AutoFit . . . 241
2.7.4.7.2 DataImportProperties.AutoFormatter . . . 242
2.7.4.7.3 DataImportProperties.ColumnIndexFilter . . . 242
2.7.4.7.4 DataImportProperties.ColumnNameFilter . . . 243
2.7.4.7.5 DataImportProperties.MaxColumns . . . 243
2.7.4.7.6 DataImportProperties.MaxRows . . . 244
2.7.4.7.7 DataImportProperties.UseColumnNames . . . 245
2.7.4.8 Document . . . 245
2.7.4.8.1 Document.Append(Document) . . . 247
2.7.4.8.2 Document.Autoformat . . . 248
2.7.4.8.3 Document.AutoformatType . . . 249
2.7.4.8.4 Document.ChangeTracking . . . 249
2.7.4.8.5 Document.CompatibilitySettings . . . 250
2.7.4.8.6 Document.CreateDataImportProperties() . . . 250
2.7.4.8.7 Document.CreateFont() . . . 251
2.7.4.8.8 Document.CreateParagraphFormatting() . . . 251
2.7.4.8.9 Document.CreateTableFormatting() . . . 252
2.7.4.8.10 Document.DateCreated . . . 252
2.7.4.8.11 Document.DatePrinted . . . 253
2.7.4.8.12 Document.DateRevised . . . 253
2.7.4.8.13 Document.DefaultTabWidth . . . 254
2.7.4.8.14 Document.DocumentProperties . . . 254
2.7.4.8.15 Document.DocumentProtection . . . 255
2.7.4.8.16 Document.GetBookmark(String) . . . 255
2.7.4.8.17 Document.GetVariable(String) . . . 256
2.7.4.8.18 Document.Gutter . . . 256
2.7.4.8.19 Document.GutterPosition . . . 257
2.7.4.8.20 Document.HyphenSettings . . . 257
2.7.4.8.21 Document.MergeStylesFromTemplate . . . 258
2.7.4.8.22 Document.NumCharacters . . . 259
2.7.4.8.23 Document.NumPagesOnLastOpen . . . 259
2.7.4.8.24 Document.NumParagraphs . . . 259
2.7.4.8.25 Document.NumWords . . . 260
2.7.4.8.26 Document.PrintDataInFormFields . . . 260
2.7.4.8.27 Document.PrintFacingPages . . . 261
2.7.4.8.28 Document.SaveDataInFormField . . . 262
2.7.4.8.29 Document.Sections . . . 262
2.7.4.8.30 Document.SetVariable(String, String) . . . 263
2.7.4.8.31 Document.Styles . . . 263
2.7.4.8.32 Document.UseWidowControl . . . 264
2.7.4.8.33 Document.ViewSettings . . . 264
2.7.4.8.34 Document.Watermark . . . 265
2.7.4.9 DocumentProperties . . . 265
2.7.4.9.1 DocumentProperties.Application . . . 267
2.7.4.9.2 DocumentProperties.Author . . . 267
2.7.4.9.3 DocumentProperties.BuiltInProperty . . . 268
2.7.4.9.4 DocumentProperties.Clear() . . . 268
2.7.4.9.5 DocumentProperties.Comments . . . 269
2.7.4.9.6 DocumentProperties.Company . . . 269
2.7.4.9.7 DocumentProperties.CreationDate . . . 270
2.7.4.9.8 DocumentProperties.GetCustomProperty(String) . . . 271
2.7.4.9.9 DocumentProperties.GetProperty . . . 271
2.7.4.9.10 DocumentProperties.Keywords . . . 273
2.7.4.9.11 DocumentProperties.LastSavedDate . . . 273
2.7.4.9.12 DocumentProperties.Manager . . . 274
2.7.4.9.13 DocumentProperties.RestoreDefaults() . . . 275
2.7.4.9.14 DocumentProperties.SetCustomProperty(String, Object) . . . 275
2.7.4.9.15 DocumentProperties.SetProperty . . . 276
2.7.4.9.16 DocumentProperties.Subject . . . 277
2.7.4.9.17 DocumentProperties.Title . . . 278
2.7.4.10 DocumentProtection . . . 278
2.7.4.10.1 DocumentProtection.AnnotationsLocked . . . 279
2.7.4.10.2 DocumentProtection.AutoFormatOverride . . . 280

2.7.4.10.3 DocumentProtection.DocumentProtected . . . 281
2.7.4.10.4 DocumentProtection.EnforceDocumentProtection . . . 281
2.7.4.10.5 DocumentProtection.ProtectionLevel . . . 282
2.7.4.10.6 DocumentProtection.SelectOnlyFormFields . . . 282
2.7.4.10.7 DocumentProtection.StylesLocked . . . 283
2.7.4.11 Element . . . 283
2.7.4.11.1 Element.BreakType . . . 286
2.7.4.11.2 Element.Children . . . 287
2.7.4.11.3 Element.CreateBookmark(String) . . . 287
2.7.4.11.4 Element.CreateSectionAfter() . . . 288
2.7.4.11.5 Element.CreateSectionBefore() . . . 288
2.7.4.11.6 Element.DeleteElement() . . . 289
2.7.4.11.7 Element.ElementType . . . 289
2.7.4.11.8 Element.GetElements(Element.Type) . . . 289
2.7.4.11.9 Element.GetPosition(Int32) . . . 290
2.7.4.11.10 Element.ImportData . . . 291
2.7.4.11.11 Element.InsertAfter(Element) . . . 297
2.7.4.11.12 Element.InsertBefore(Element) . . . 298
2.7.4.11.13 Element.InsertBreakAfter(Element.BreakType) . . . 298
2.7.4.11.14 Element.InsertBreakBefore(Element.BreakType) . . . 298
2.7.4.11.15 Element.InsertHyperlinkAfter(String, String) . . . 299
2.7.4.11.16 Element.InsertHyperlinkBefore(String, String) . . . 299
2.7.4.11.17 Element.InsertImageAfter . . . 300
2.7.4.11.18 Element.InsertImageBefore . . . 301
2.7.4.11.19 Element.InsertListAfter(Boolean) . . . 303
2.7.4.11.20 Element.InsertListBefore(Boolean) . . . 303
2.7.4.11.21 Element.InsertMergeFieldAfter(String, String) . . . 304
2.7.4.11.22 Element.InsertMergeFieldBefore(String, String) . . . 304
2.7.4.11.23 Element.InsertParagraphAfter . . . 305
2.7.4.11.24 Element.InsertParagraphBefore . . . 307
2.7.4.11.25 Element.InsertTableAfter . . . 308
2.7.4.11.26 Element.InsertTableBefore . . . 310
2.7.4.11.27 Element.InsertTextAfter . . . 311
2.7.4.11.28 Element.InsertTextBefore . . . 313
2.7.4.11.29 Element.Length . . . 314
2.7.4.11.30 Element.Search(String) . . . 314
2.7.4.11.31 Element.SearchAndReplace(String, String) . . . 315
2.7.4.11.32 Element.Start . . . 316
2.7.4.11.33 Element.SubElement(Int32, Int32) . . . 317
2.7.4.11.34 Element.Text . . . 317
2.7.4.11.35 Element.Type . . . 318
2.7.4.12 Field . . . 319
2.7.4.12.1 Field.Code . . . 320
2.7.4.12.2 Field.Contents . . . 320
2.7.4.12.3 Field.FieldType . . . 320
2.7.4.12.4 Field.Type . . . 323
2.7.4.12.5 Field.Update() . . . 324
2.7.4.13 Font . . . 324
2.7.4.13.1 Font.Animation . . . 326
2.7.4.13.2 Font.Bold . . . 327
2.7.4.13.3 Font.Border . . . 327
2.7.4.13.4 Font.Caps . . . 328
2.7.4.13.5 Font.CharacterScaling . . . 328
2.7.4.13.6 Font.CharacterSpacing . . . 329
2.7.4.13.7 Font.DoubleStrike . . . 330
2.7.4.13.8 Font.Embossed . . . 330
2.7.4.13.9 Font.Engraved . . . 331
2.7.4.13.10 Font.FontName . . . 332
2.7.4.13.11 Font.FontSize . . . 332
2.7.4.13.12 Font.Hidden . . . 333
2.7.4.13.13 Font.Highlight . . . 333
2.7.4.13.14 Font.Hyphenation . . . 334
2.7.4.13.15 Font.HyphenationRule . . . 335
2.7.4.13.16 Font.Italic . . . 335
2.7.4.13.17 Font.Kerning . . . 336
2.7.4.13.18 Font.NoProofChecking . . . 337
2.7.4.13.19 Font.Outlined . . . 337
2.7.4.13.20 Font.ScriptType . . . 338
2.7.4.13.21 Font.Shading . . . 338
2.7.4.13.22 Font.Shadow . . . 338
2.7.4.13.23 Font.SmallCaps . . . 339
2.7.4.13.24 Font.Strike . . . 340
2.7.4.13.25 Font.SuperSubScript . . . 340
2.7.4.13.26 Font.TextAnimation . . . 341
2.7.4.13.27 Font.TextColor . . . 342

2.7.4.13.28 Font.Underline . . . 342
2.7.4.13.29 Font.UnderlineColor . . . 343
2.7.4.13.30 Font.UnderlineType . . . 343
2.7.4.13.31 Font.VerticalPosition . . . 345
2.7.4.14 Hyperlink . . . 345
2.7.4.14.1 Hyperlink.GetUrlString() . . . 346
2.7.4.15 InlineImage . . . 347
2.7.4.15.1 InlineImage.Format . . . 348
2.7.4.15.2 InlineImage.GetImage(InlineImage.Format) . . . 348
2.7.4.15.3 InlineImage.Height . . . 349
2.7.4.15.4 InlineImage.Image . . . 350
2.7.4.15.5 InlineImage.ImageFormat . . . 350
2.7.4.15.6 InlineImage.Width . . . 351
2.7.4.16 List . . . 351
2.7.4.16.1 List.AddEntry(Int32) . . . 352
2.7.4.16.2 List.GetEntry(Int32) . . . 353
2.7.4.16.3 List.GetLevel(Int32) . . . 354
2.7.4.16.4 List.InsertEntry(Int32, Int32) . . . 354
2.7.4.16.5 List.NumEntries . . . 355
2.7.4.17 ListEntry . . . 355
2.7.4.17.1 ListEntry.BulletText . . . 356
2.7.4.17.2 ListEntry.LevelNum . . . 357
2.7.4.17.3 ListEntry.RestartNumbering() . . . 357
2.7.4.18 ListLevel . . . 357
2.7.4.18.1 ListLevel.BulletText . . . 358
2.7.4.18.2 ListLevel.Indent . . . 359
2.7.4.18.3 ListLevel.NumberFormat . . . 360
2.7.4.18.4 ListLevel.NumberIndent . . . 360
2.7.4.19 MergeField . . . 361
2.7.4.19.1 MergeField.GetFieldName() . . . 362
2.7.4.20 NamedStyle . . . 362
2.7.4.20.1 NamedStyle.BuiltIn . . . 363
2.7.4.20.2 NamedStyle.BuiltInStyleName . . . 367
2.7.4.20.3 NamedStyle.Font . . . 368
2.7.4.20.4 NamedStyle.Name . . . 368
2.7.4.20.5 NamedStyle.ParagraphFormatting . . . 369
2.7.4.20.6 NamedStyle.StyleType . . . 369
2.7.4.20.7 NamedStyle.TableFormatting . . . 370
2.7.4.20.8 NamedStyle.Type . . . 370
2.7.4.21 Paragraph . . . 371
2.7.4.21.1 Paragraph.Formatting . . . 372
2.7.4.21.2 Paragraph.Style . . . 372
2.7.4.22 ParagraphFormatting . . . 373
2.7.4.22.1 ParagraphFormatting.AbsolutePositioning . . . 375
2.7.4.22.2 ParagraphFormatting.AfterAutoSpacing . . . 376
2.7.4.22.3 ParagraphFormatting.AllowAutoHyphenation . . . 376
2.7.4.22.4 ParagraphFormatting.BeforeAutoSpacing . . . 377
2.7.4.22.5 ParagraphFormatting.FontAlignment . . . 378
2.7.4.22.6 ParagraphFormatting.GetBorder(Border.Location) . . . 378
2.7.4.22.7 ParagraphFormatting.GetIndent(ParagraphFormatting.IndentLocation) . . . 379
2.7.4.22.8 ParagraphFormatting.IndentLocation . . . 380
2.7.4.22.9 ParagraphFormatting.Justification . . . 380
2.7.4.22.10 ParagraphFormatting.KeepLinesTogether . . . 380
2.7.4.22.11 ParagraphFormatting.KeepWithNext . . . 381
2.7.4.22.12 ParagraphFormatting.LineNumberingAllowed . . . 382
2.7.4.22.13 ParagraphFormatting.LineSpacing . . . 382
2.7.4.22.14 ParagraphFormatting.LineSpacingRule . . . 383
2.7.4.22.15 ParagraphFormatting.PageBreakBefore . . . 383
2.7.4.22.16 ParagraphFormatting.SetIndent(Int32, ParagraphFormatting.IndentLocation) . . . 384
2.7.4.22.17 ParagraphFormatting.SetLineSpacing(Int32, ParagraphFormatting.SpacingRule) . . . 384
2.7.4.22.18 ParagraphFormatting.Shading . . . 385
2.7.4.22.19 ParagraphFormatting.SpaceAfter . . . 386
2.7.4.22.20 ParagraphFormatting.SpaceBefore . . . 386
2.7.4.22.21 ParagraphFormatting.SpacingRule . . . 387
2.7.4.22.22 ParagraphFormatting.TextJustification . . . 387
2.7.4.22.23 ParagraphFormatting.WidowControl . . . 388
2.7.4.23 Position . . . 389
2.7.4.23.1 Position.InsertHyperlink(String, String) . . . 390
2.7.4.23.2 Position.InsertImage . . . 390
2.7.4.23.3 Position.InsertList(Boolean) . . . 392
2.7.4.23.4 Position.InsertMergeField(String, String) . . . 392
2.7.4.23.5 Position.InsertParagraph(NamedStyle) . . . 393
2.7.4.23.6 Position.InsertSection() . . . 393
2.7.4.23.7 Position.InsertTable(Int32, Int32) . . . 394
2.7.4.23.8 Position.InsertText(String, Boolean) . . . 395

2.7.4.23.9 Position.PositionValue . . . 395
2.7.4.24 SearchMatch . . . 396
2.7.4.24.1 SearchMatch.Element . . . 397
2.7.4.24.2 SearchMatch.Group(Int32) . . . 397
2.7.4.24.3 SearchMatch.GroupCount . . . 398
2.7.4.24.4 SearchMatch.Replace(String) . . . 399
2.7.4.25 Section . . . 399
2.7.4.25.1 Section.BorderAlwaysInFront . . . 402
2.7.4.25.2 Section.BorderAppliedTo . . . 402
2.7.4.25.3 Section.BorderArea . . . 402
2.7.4.25.4 Section.BorderOffsetsFromPage . . . 403
2.7.4.25.5 Section.BottomMargin . . . 403
2.7.4.25.6 Section.Break . . . 403
2.7.4.25.7 Section.BreakType . . . 404
2.7.4.25.8 Section.FirstLineNum . . . 404
2.7.4.25.9 Section.FirstPageNum . . . 405
2.7.4.25.10 Section.FooterY . . . 406
2.7.4.25.11 Section.GetBorder(Border.Location) . . . 406
2.7.4.25.12 Section.GetFooter(Section.HeaderFooterType) . . . 407
2.7.4.25.13 Section.GetHeader(Section.HeaderFooterType) . . . 408
2.7.4.25.14 Section.GutterWidth . . . 408
2.7.4.25.15 Section.HasTitlePage . . . 409
2.7.4.25.16 Section.HeaderFooterType . . . 409
2.7.4.25.17 Section.HeaderY . . . 410
2.7.4.25.18 Section.LeftMargin . . . 410
2.7.4.25.19 Section.LineNumbering . . . 411
2.7.4.25.20 Section.LineNumberingType . . . 411
2.7.4.25.21 Section.LineNumberX . . . 412
2.7.4.25.22 Section.LineNumModulus . . . 413
2.7.4.25.23 Section.LinesBetween . . . 413
2.7.4.25.24 Section.NumberFormat . . . 414
2.7.4.25.25 Section.NumColumns . . . 414
2.7.4.25.26 Section.Orientation . . . 415
2.7.4.25.27 Section.PageHeight . . . 415
2.7.4.25.28 Section.PageNumberFormat . . . 416
2.7.4.25.29 Section.PageOrientation . . . 417
2.7.4.25.30 Section.PagePaperSize . . . 417
2.7.4.25.31 Section.PageSize . . . 419
2.7.4.25.32 Section.PageWidth . . . 419
2.7.4.25.33 Section.RestartPageNum . . . 420
2.7.4.25.34 Section.RightMargin . . . 420
2.7.4.25.35 Section.TopMargin . . . 421
2.7.4.25.36 Section.Unlocked . . . 422
2.7.4.26 Shading . . . 422
2.7.4.26.1 Shading.BackgroundColor . . . 423
2.7.4.26.2 Shading.ForegroundColor . . . 423
2.7.4.26.3 Shading.Pattern . . . 424
2.7.4.26.4 Shading.ShadingPattern . . . 425
2.7.4.27 Styles . . . 426
2.7.4.27.1 Styles.Item . . . 427
2.7.4.27.2 Styles.Length() . . . 429
2.7.4.28 Table . . . 430
2.7.4.28.1 Table.AbsolutePositioning . . . 432
2.7.4.28.2 Table.AddColumns(Int32) . . . 432
2.7.4.28.3 Table.AddRows(Int32) . . . 433
2.7.4.28.4 Table.CreateBookmarkOnRow(Int32, String) . . . 433
2.7.4.28.5 Table.DeleteColumns(Int32, Int32) . . . 434
2.7.4.28.6 Table.DeleteRows(Int32, Int32) . . . 434
2.7.4.28.7 Table.Formatting . . . 435
2.7.4.28.8 Table.GetNumColumns(Int32) . . . 435
2.7.4.28.9 Table.GetPreferredColumnWidth(Int32) . . . 436
2.7.4.28.10 Table.GetPreferredColumnWidthUnits(Int32) . . . 437
2.7.4.28.11 Table.GetPreferredWidth(out Units) . . . 437
2.7.4.28.12 Table.GetRowHeight(Int32) . . . 438
2.7.4.28.13 Table.GetRowHeightExact(Int32) . . . 439
2.7.4.28.14 Table.ImportDataRow(Object()) . . . 439
2.7.4.28.15 Table.InsertColumns(Int32, Int32) . . . 440
2.7.4.28.16 Table.InsertRows(Int32, Int32) . . . 440
2.7.4.28.17 Table.Item(Int32, Int32) . . . 441
2.7.4.28.18 Table.MergeCells(Int32, Int32, Int32, Int32) . . . 442
2.7.4.28.19 Table.NumRows . . . 442
2.7.4.28.20 Table.PreferredWidth . . . 443
2.7.4.28.21 Table.SetPreferredColumnWidth(Int32, Int32) . . . 444
2.7.4.28.22 Table.SetPreferredColumnWidthUnits(Int32, TableCell.WidthUnits) . . . 444
2.7.4.28.23 Table.SetPreferredWidth(Double, Units) . . . 445

2.7.4.28.24 Table.SetRowHeight(Int32, Int32) . . . 446
2.7.4.28.25 Table.SetRowHeightExact(Int32, Boolean) . . . 447
2.7.4.28.26 Table.Style . . . 447
2.7.4.29 TableCell . . . 448
2.7.4.29.1 TableCell.ApplyFont(Font) . . . 449
2.7.4.29.2 TableCell.ApplyParagraphFormatting(ParagraphFormatting) . . . 450
2.7.4.29.3 TableCell.CellWidthUnits . . . 450
2.7.4.29.4 TableCell.FirstMerged . . . 451
2.7.4.29.5 TableCell.FirstVerticalMerged . . . 452
2.7.4.29.6 TableCell.GetBorder(Border.Location) . . . 452
2.7.4.29.7 TableCell.GetPadding(TableCell.Location) . . . 453
2.7.4.29.8 TableCell.Location . . . 454
2.7.4.29.9 TableCell.Merged . . . 454
2.7.4.29.10 TableCell.NoWrap . . . 455
2.7.4.29.11 TableCell.PreferredWidth . . . 455
2.7.4.29.12 TableCell.SetPadding(TableCell.Location, Int32) . . . 456
2.7.4.29.13 TableCell.Shading . . . 457
2.7.4.29.14 TableCell.VertAlignment . . . 457
2.7.4.29.15 TableCell.VerticalAlignment . . . 458
2.7.4.29.16 TableCell.VerticalMerged . . . 458
2.7.4.29.17 TableCell.WidthUnits . . . 459
2.7.4.30 TableFormatting . . . 459
2.7.4.30.1 TableFormatting.AutoWidth . . . 461
2.7.4.30.2 TableFormatting.CantSplit . . . 461
2.7.4.30.3 TableFormatting.DefaultShading . . . 462
2.7.4.30.4 TableFormatting.DefaultSpacing . . . 462
2.7.4.30.5 TableFormatting.GetDefaultBorder(Border.Location) . . . 463
2.7.4.30.6 TableFormatting.GetDefaultPadding(TableCell.Location) . . . 464
2.7.4.30.7 TableFormatting.Justification . . . 464
2.7.4.30.8 TableFormatting.LeftIndent . . . 465
2.7.4.30.9 TableFormatting.RepeatAsHeader . . . 466
2.7.4.30.10 TableFormatting.SetDefaultPadding(TableCell.Location, Int32) . . . 466
2.7.4.31 TwipsConverter . . . 467
2.7.4.31.1 TwipsConverter.FromCentimeters(Double) . . . 468
2.7.4.31.2 TwipsConverter.FromInches(Double) . . . 468
2.7.4.31.3 TwipsConverter.FromPoints(Double) . . . 469
2.7.4.31.4 TwipsConverter.ToCentimeters(Int32) . . . 470
2.7.4.31.5 TwipsConverter.ToInches(Int32) . . . 470
2.7.4.31.6 TwipsConverter.ToPoints(Int32) . . . 471
2.7.4.32 Units . . . 472
2.7.4.33 Watermark . . . 472
2.7.4.33.1 Watermark.ClearWatermark() . . . 473
2.7.4.33.2 Watermark.Color . . . 473
2.7.4.33.3 Watermark.FontName . . . 474
2.7.4.33.4 Watermark.Image . . . 475
2.7.4.33.5 Watermark.Layout . . . 475
2.7.4.33.6 Watermark.Scale . . . 476
2.7.4.33.7 Watermark.Semitransparent . . . 476
2.7.4.33.8 Watermark.Size . . . 477
2.7.4.33.9 Watermark.Text . . . 478
2.7.4.33.10 Watermark.Type . . . 478
2.7.4.33.11 Watermark.Washout . . . 479
2.7.4.33.12 Watermark.WatermarkLayout . . . 479
2.7.4.33.13 Watermark.WatermarkType . . . 480
2.7.4.34 WordApplication . . . 480
2.7.4.34.1 WordApplication.AlwaysUseUnicode . . . 482
2.7.4.34.2 WordApplication.Create() . . . 483
2.7.4.34.3 WordApplication.LicenseKey . . . 483
2.7.4.34.4 WordApplication.Open . . . 484
2.7.4.34.5 WordApplication.Preserve . . . 486
2.7.4.34.6 WordApplication.Save . . . 487
2.7.4.34.7 WordApplication.Version . . . 491
2.7.4.34.8 WordApplication.WordApplication() . . . 491
2.7.4.35 WordChangeTracking . . . 492
2.7.4.35.1 WordChangeTracking.DisplayRevisions . . . 493
2.7.4.35.2 WordChangeTracking.LockRevisionMarkingState . . . 493
2.7.4.35.3 WordChangeTracking.PrintRevisions . . . 494
2.7.4.35.4 WordChangeTracking.ShowAnnotationRevisions . . . 494
2.7.4.35.5 WordChangeTracking.ShowFormattingRevisions . . . 495
2.7.4.35.6 WordChangeTracking.ShowInkRevisions . . . 496
2.7.4.35.7 WordChangeTracking.ShowInsertDeleteRevisions . . . 496
2.7.4.35.8 WordChangeTracking.ShowMarkupRevisions . . . 497
2.7.4.35.9 WordChangeTracking.TrackChanges . . . 497
2.7.4.36 WordHyphenSettings . . . 498
2.7.4.36.1 WordHyphenSettings.AllowedConsecutiveHyphens . . . 499

2.7.4.36.2 WordHyphenSettings.AutoHyphen . . . 500
2.7.4.36.3 WordHyphenSettings.CapitalHyphenation . . . 500
2.7.4.36.4 WordHyphenSettings.HyphenationWidth . . . 501
2.7.4.37 WordNumberFormat . . . 502
2.7.4.38 WordViewSettings . . . 503
2.7.4.38.1 WordViewSettings.DisplayedOutlineLevels . . . 505
2.7.4.38.2 WordViewSettings.DisplayFieldResults . . . 505
2.7.4.38.3 WordViewSettings.DisplayHidden . . . 506
2.7.4.38.4 WordViewSettings.DocumentView . . . 507
2.7.4.38.5 WordViewSettings.EnvelopeVisible . . . 507
2.7.4.38.6 WordViewSettings.IncludeFooterInBorder . . . 508
2.7.4.38.7 WordViewSettings.IncludeHeaderInBorder . . . 509
2.7.4.38.8 WordViewSettings.ShadeFormFields . . . 509
2.7.4.38.9 WordViewSettings.ViewType . . . 510
2.7.4.38.10 WordViewSettings.Zoom . . . 510
2.7.4.38.11 WordViewSettings.ZoomPercentage . . . 511
2.7.4.38.12 WordViewSettings.ZoomType . . . 512
2.7.4.39 WordWriterException . . . 512
2.7.5 SharePointIntegration Programmer's Reference . . . 513
2.7.5.1 SharePointIntegration . . . 513
2.7.5.1.1 SharePointIntegration.ImportData(Element, Microsoft.SharePoint.SPList) . . . 514
2.7.5.1.2 SharePointIntegration.ImportData(Element, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties) . . . 515
2.7.5.1.3 SharePointIntegration.Open(WordApplication, Microsoft.SharePoint.SPDocumentLibrary, String) . . . 516
2.7.5.1.4 SharePointIntegration.Open(WordApplication, Microsoft.SharePoint.SPListItem, String) . . . 517
2.7.5.1.5 SharePointIntegration.Open(WordTemplate, Microsoft.SharePoint.SPDocumentLibrary, String) . . . 518
2.7.5.1.6 SharePointIntegration.Open(WordTemplate, Microsoft.SharePoint.SPListItem, String) . . . 519
2.7.5.1.7 SharePointIntegration.Save(WordApplication, Document, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean) . . . 520
2.7.5.1.8 SharePointIntegration.Save(WordApplication, Document, Microsoft.SharePoint.SPListItem, String) . . . 521
2.7.5.1.9 SharePointIntegration.Save(WordTemplate, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean) . . . 523
2.7.5.1.10 SharePointIntegration.Save(WordTemplate, Microsoft.SharePoint.SPListItem, String) . . . 524
2.7.5.1.11 SharePointIntegration.SetDataSource(WordTemplate, Microsoft.SharePoint.SPList, String) . . . 525
2.7.5.1.12 SharePointIntegration.SetDataSource(WordTemplate, Microsoft.SharePoint.SPList) . . . 526
2.7.5.1.13 SharePointIntegration.SetDataSource(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String) . . . 527
2.7.5.1.14 SharePointIntegration.SetDataSource(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList) . . . 528
2.7.5.1.15 SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPList, Int32) . . . 530
2.7.5.1.16 SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPList) . . . 531
2.7.5.1.17 SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, Int32) . . . 532
2.7.5.1.18 SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList) . . . 533
2.7.5.1.19 SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPList, String, Int32) . . . 534
2.7.5.1.20 SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPList, String) . . . 536
2.7.5.1.21 SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, Int32) . . . 537
2.7.5.1.22 SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String) . . . 538
2.8 Installation . . . 540
2.8.1 Installing WordWriter . . . 540
2.8.2 ASP Installation . . . 541
2.8.3 Removing WordWriter . . . 542
2.8.4 Security Considerations . . . 544
2.9 HTMLToWord Object . . . 545
2.9.1 Using HTMLToWord . . . 545
2.9.2 HTML tags supported by HTMLToWord . . . 549
2.9.3 HTMLToWord Programmer's Reference . . . 551
2.9.3.1 HTMLToWord . . . 551
2.9.3.1.1 HTMLToWord.DebugLogLevel . . . 553
2.9.3.1.2 HTMLToWord.FormatDelegate . . . 554
2.9.3.1.3 HTMLToWord.FormatElementDelegate . . . 555
2.9.3.1.4 HTMLToWord.GetImageDelegate . . . 556
2.9.3.1.5 HTMLToWord.HTMLInsertProperties . . . 556
2.9.3.1.6 HTMLToWord.HTMLInsertProperties.AcceptUnknownTags . . . 556
2.9.3.1.7 HTMLToWord.HTMLInsertProperties.DefaultFont . . . 557
2.9.3.1.8 HTMLToWord.HTMLInsertProperties.IncludeContentsOfUnknownTags . . . 558
2.9.3.1.9 HTMLToWord.HTMLTagAction . . . 559
2.9.3.1.10 HTMLToWord.ImageDelegate . . . 559
2.9.3.1.11 HTMLToWord.InsertDelegate . . . 561
2.9.3.1.12 HTMLToWord.InsertElementDelegate . . . 563
2.9.3.1.13 HTMLToWord.InsertHTML . . . 563
2.9.3.1.14 HTMLToWord.InsertProperties . . . 567
2.9.3.1.15 HTMLToWord.LogFileName . . . 568

2.9.3.1.16 HTMLToWord.LoggingLevel 568
2.9.4 The HTMLToWord Object 569
2.10 OfficeWriter Links 570
2.11 External Links 571
3. WordTemplate 2003 feature limitation 571

Documentation

WordWriter for Windows Version 4

- **Introduction**
  describes the basic features and requirements of OfficeWriter for Word.
- **Quickstart**
  will create your first Word document in minutes. This brief tutorial provides a high level overview of creating Word documents with OfficeWriter.
- **WordTemplate in Depth**
  explains template-driven document creation in detail.
- **WordApplication in Depth**
  introduces each major area of OfficeWriter's powerful new WordApplication object, which allows you to generate a document without a template and modify every aspect of it at run-time.
- **SharePoint Integration in Depth**
  introduces new extension methods for OfficeWriter objects for use in SharePoint Applications.
- **WordWriter in ASP**
  Describes how to access WordWriter from ASP using the COM-callable wrapper.
- **Programmer's Reference**
  describes each object, method, and property.
- **Installation**
  includes instructions for automatic and manual installation. System Administrators will find useful information here.
- **HTMLToWord Object**
  Describes how to use the open source HTMLToWord object with WordWriter to convert XHTML content to a Word document.
- **OfficeWriter Links**
  Links to the other parts of OfficeWriter
- **External Links**
  Links to SoftArtisans resources outside of the documentation that can help you get up and running with OfficeWriter

Introduction

- What is WordWriter?
- Features and Benefits
- WordWriter Editions
- New Features
- Requirements
- Frequently Asked Questions
- Office 2007 Support
- Change Log

What is WordWriter?

WordWriter is a high performance .NET assembly that creates Microsoft Word documents on a Web server. A few simple lines of code generate editable presentation-quality documents that can be saved or viewed instantly by thousands of concurrent users.

The documents generated by WordWriter are in Microsoft Word format (.doc, .dot) or Office Open XML format (.docx, .docm, .dotx, .dotm) and preserve all of the familiar features associated with Word documents. WordWriter offers high-speed performance and, since Microsoft Word is not required on the Web server running WordWriter, your Web applications have licensing flexibility and ease of deployment.

Using WordWriter, you can generate a document from an existing Word file, a WordWriter template, or generate a Word file from code:

Using a Template	***Maximum Speed*** To generate documents with the greatest speed: 1. In Microsoft Word, create a template file containing merge fields that will bind to one or more data sources. 2. In code, use WordWriter to open the template, set data sources, populate merge fields, and save the new Word file. Files can be saved to disk, streamed to the client, or both.
Using an Existing Document	***Increased Flexibility*** When you modify an existing document using WordWriter code, all Word features and functionality are preserved in the newly generated document. You can easily enhance documents, at runtime, with WordWriter's extensive design capabilities.
Pure Coding	***Completely Dynamic*** This dynamic approach provides total control, at runtime, of every element in the document.

Features and Benefits

- **Flexible Report Creation:**
  WordWriter generates native Word documents from existing .doc files, templates, or code.
- **Development Options:**
  Choice of code-based or template-driven development.
- **Real Word:**
  WordWriter generates Word binary and Office Open XML files.
- **High Performance:**
  Built for the most demanding sites, WordWriter easily generates over 100,000 documents in an hour.
- **Word Not Required:**
  Since Word is not needed on the server, WordWriter is perfect for demanding Web sites. See why Microsoft recommends keeping Office off the server.
- **Delivery Methods:**
  Save the generated document to disk, stream it to a Web browser, or both.

WordWriter Editions

Production Editions

- **WordWriterEE** is for enterprise-class Web applications. WordWriterEE includes support for both code-based and template-driven development, Reporting Services integration and Nesting and Grouping.
- **WordWriterSE** supports template-driven development and limited Reporting Services integration. WordWriterSE does not support code-based development with WordApplication.
- **OfficeWriter**
  OfficeWriter unites WordWriter and ExcelWriter, allowing users to create, share, and analyze crucial business information through both powerful spreadsheets and rich documents.

Development Editions

- **OfficeWriterEE Developer** offers the same functionality as OfficeWriterEE (which combines ExcelWriterEE and WordWriterEE), but can generate only one report at a time, which is typical of the development environment. Other restrictions may apply.
- **OfficeWriterSE Developer** offers the same functionality as OfficeWriterSE (which combines ExcelWriterSE and WordWriterSE), but can generate only one report at a time. Other restrictions may apply.

Licensing

WordWriter licensing is per CPU. A license is required for each CPU where WordWriter is installed. There are no per user, per IP address, per Web site, or per server licenses. The WordWriter license does not allow redistribution.

Compare WordWriter Editions

Feature	WordWriterEE	WordWriterSE
Template-Driven Development	✔	✔
Pure Code-based Development	✔	---
Nesting and Grouping Functionality	✔	---
Reporting Services Integration	✔	Limited
OfficeWriter Assistant (file-transfer tool)	✔	---
Documents Delivered Per Day	Unlimited	Unlimited

New Features

New in Version 4.6

- **WordTemplate COM-callable wrapper is no longer automatically installed**
  - COM-callable wrapper is available upon request
  - Since it is no longer needed in the default installation, WordWriter is no longer installed in the GAC by the automatic installer

New in Version 4.5

- **SharePoint Integration**
  - WordWriter now offers SharePoint Integration extension methods using the new SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll. By including the new DLL as a reference to your application page within a SharePoint application, you can use SharePoint Lists and Document Libraries as data sources and save documents as SharePoint List attachments or Document Library Items. For more details see Exporting list as Word document.

- **Grouping and Nesting**
  - WordTemplate Enterprise Edition now supports grouping and nesting for OOXML (.docx, .docm, .dotx, .dotm) templates through the use of special bookmarks arranged in grouping blocks. For more details see Creating a Grouped and Nested Document.

- **Document.Append Method**
  - The Document.Append method allows you to append one document to another, preserving the formatting, header, and footers of both documents.

- **WordTemplate.SaveExtension Property**
  - The read-only WordTemplate.SaveExtension property returns the extension, sans point, of the document that has been opened by WordTemplate.

New in Version 4.1

- **WordTemplate Formulas**
  - WordTemplate now supports embedded formulas in Word 2007 OOXML file format files (.docx/.docm) that are evaluated server-side based on values from data source fields. For more details, see Creating Merge Fields and Adding Formulas to a Document.

New in Version 4.0

- **Office Open XML File Support**
  - WordTemplate now supports the Office Open XML (OOXML) format. OOXML is the default file format for Microsoft Office 2007 products, including Excel and Word. The file format consists of XML-based files and binary files compressed in a ZIP file. This results in smaller files than the binary format used by previous versions of Excel and Word. The OOXML format is defined in the standards documents Ecma-376 and ISO/DIS 29500. Word documents in the OOXML format use the .docm file extension if they contain embedded macros, and the .docx file extension if they do not. Note that WordApplication does not support the OOXML format.

- **No Dependence on the J# Redistributable**
  - OfficeWriter version 4 is now completely implemented using only C#. This negates the previous system requirement to install the Microsoft J# redistributable in order to use OfficeWriter

- **Performance Improvements**
  - Processing large binary Word files is up to 6 times faster with WordApplication and 35% faster with WordTemplate
  - Processing large files or importing data with WordTemplate is 40 times more memory efficient with OOXML files (.docx and .docm)

- **Ability to preserve merge fields**
  - New PreserveFields property on the WordTemplate object will keep merge fields in the output file rather than replacing them with values.

Requirements

Server Requirements

- Windows 2000, XP, 2003 Server, 2003 Server R2, Vista, 7, 2008 Server, or 2008 Server R2. 64-bit Operating Systems are supported.
- Microsoft .NET Framework Redistributable version 2.0 or later is required.

Server Requirements for Reporting Services Integration

To use WordWriter with Reporting Services 2005, you will need:

- Windows 2000, XP, 2003 Server, Vista, or 2008 Server. 64-bit Operating Systems are supported.
- MS Reporting Services
- Microsoft .NET Framework Redistributable version 2.0 or later.
- To run the Reporting Services samples, SQL Server 2005 and the AdventureWorks database must be installed.

To use WordWriter with Reporting Services 2008, you will need:

- Windows 2000, XP, 2003 Server, Vista, or 2008 Server. 64-bit Operating Systems are supported.
- MS Reporting Services
- Microsoft .NET Framework Redistributable version 3.5 SP1 or later.
- To run the Reporting Services samples, SQL Server 2008 and the AdventureWorks database must be installed.

> Microsoft Word is not required on the server. This prevents the performance penalties and instability associated with instantiating multiple copies of the Word object on a Web server.

Server Requirements for ASP

To use the COM-callable wrapper for WordTemplate in classic ASP, you will need the Visual C++ 2005 SP1 redistributable package installed on your server. Install either the 32-bit version or 64-bit version, depending on your architecture.

Client Requirements

To create WordWriter templates, you will need:

- Microsoft Word 97, 2000, 2002 (XP), 2003, or 2007 (see Office 2007 Support).

> WordWriter templates can be created in Word 2003, however, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.

No software is required on the user's machine to download and save a Word file. To enable opening of Word files, one of the following must be installed on the client:

- Microsoft Word 97, 2000, 2002 (XP), 2003, or 2007 (see Office 2007 Support).

> To open files written in the Office OpenXML (.docx) format, you must either use Word 2007 or install the Microsoft Office Compatibility Pack for Word 2000 or greater.

- Microsoft's free Word Viewer. The Word Viewer allows the user to view and print Word files, but not to modify them.
- Another application that supports the Word format, such as, StarOffice or OpenOffice.org for Unix.

Client Requirements for Reporting Services Integration

To use WordWriter with Reporting Services, you will need:

- Microsoft Word 2000, 2002 (XP), 2003, or 2007 (see Office 2007 Support for WordWriter).
- MS Query or Visual Studio.NET

> In most cases, data sources and queries will be defined in Visual Studio .NET. MS Query is required only if you wish to allow report designers to define data sources and queries within Microsoft Word.

MS Query is part of Microsoft Office, but may not be installed by default. To install MS Query:

1. Have your Office Installation CD ready.
2. Close all programs.
3. Double-click **Add or Remove Programs** in the Control Panel.
4. Select **Microsoft Office**, and click the **Change** button.
5. Select **Add or Remove Features**.
6. Check **Choose advanced customization of applications**.
7. Expand **Office Tools**.
8. Select **Microsoft Query** and click **Update** .

Frequently Asked Questions

What software do I need on my server to run WordWriter?

What software do I need to create a WordWriter template?

What software is required on the client to view a WordWriter generated file?

What versions of Microsoft Word does WordWriter support?

Does WordWriter include features that are not available in Microsoft Word?

What data sources can WordWriter use?

Can I include more than one data source in a single WordWriter template?

Can WordWriter merge documents or modify a document's format at runtime?

What software do I need on my server to run WordWriter?

There are four primary server requirements to run WordWriter on the server:

- Windows 2000, Windows XP, Windows 2003 Server, Windows Vista, Windows 2008 Server, or Windows 7. 64-bit Operating Systems are supported.
- IIS 5 or higher
- Microsoft .NET Framework Redistributable Version 2.0 or higher

What software do I need to create a WordWriter template?

A WordWriter template is a Word document that contains merge fields. You can create a WordWriter template using Microsoft Word 97, 2000, 2002 (XP), 2003, or 2007.

> WordWriter templates can be created in Word 2003, however, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.

What software is required on the client to view a WordWriter generated file?

To open a Word file generated by WordWriter, you can use:

- Microsoft Word 97, 2000, 2002 (XP), 2003, or 2007. Office Open XML (.docx) files cannot be opened in Word 97, though they can be opened in Word 2000, 2002, or 2003 using the Microsoft Office Open XML Compatibility Pack.
- Microsoft's free Word Viewer. The Word Viewer allows the user to view and print Word files, but not to modify them.
- Another application that supports the Word format, such as, StarOffice or OpenOffice for Unix, Windows, MacIntosh, or Linux.

What versions of Microsoft Word does WordWriter support?

WordWriter supports Microsoft Word 97, 2000, 2002 (XP), 2003, and 2007. Word 2007's default file format, Office Open XML (.docx), can only be used with WordTemplate and is not yet available from the WordApplication object.

> WordWriter templates can be created in Word 2003, however, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.

Does WordWriter include features that are not available in Microsoft Word?

Yes. Unlike Microsoft Word, WordWriter can stream a Word file to the browser.

What data sources can WordWriter use?

In ASP.NET, WordWriter can populate data into a Word document from a DataSet, a DataTable, an SqlDataReader, an OleDbDataReader, an AdomdDataReader, a 1-dimensional array, a rectangular array, or a jagged array. Using the SharePointIntegration extension methods, WordWriter can also populate data into a Word document from a SharePoint List or a SharePoint View.

In ASP, WordWriter can populate a Word file from a Recordset, a 1-dimensional array, or a 2-dimensional array.

Can I include more than one data source in a single WordWriter template?

Yes. To populate a template from more than one data source, or to import multiple rows from a data source, use repeat blocks. For information on creating and populating repeat blocks, see:

- Creating a Repeat Block
- Importing from Multiple Data Sources
- Importing Multiple Rows
- SetRepeatBlock Method

Can the WordTemplate object merge documents or modify a document's format at runtime?

No. At runtime, the WordTemplate can open a template Word file, populate it from data sources, and either save the new file or stream it to the client. The WordTemplate object cannot change a document's format, edit static content, or merge multiple files at runtime. For such functions, you must use the WordApplication object.

Office 2007 Support

Microsoft Office 2007 applications use a new XML-based file format instead of the binary file format used in previous versions. This release of OfficeWriter for Word supports the Office 2007 format for use with WordTemplate. (Use with WordApplication is not supported in this release.)

The file format, Office Open XML(OOXML), uses a ZIP file to package all the text files and embedded binary files that may compose a document. The text files can use any of several XML-based markup languages, such as WordprocessingML for word processing. The binary files can include formats such BMP for embedded images. Microsoft Office 2007 uses the .docm file extension for Word files containing macros, and the .docx file extension for Word files that do not.

WordTemplate supports the new Office 2007 format by opening an WordTemplate docm or docx file, populating it with data, and returning the same file type. Translation between Office 2007 files and files generated using previous releases of Office is not supported.

The OOXML standard is defined in Ecma-276 and ISO/DIS 29500.

Change Log

Below are a list of added features and bug fixes that are incorporated for each minor and major build of WordWriter.

Differences between 4.6 and 4.6.1

There were no changes to WordWriter between 4.6 and 4.6.1.

Differences between 4.5.1 and 4.6

Issues resolved

- Some files processed by WordTemplate, then WordApplication, caused errors when opened in Word 2010
- WordWriter threw an error when repeat blocks were in tables in templates created using Word 2010

Differences between 4.5 and 4.5.1

Issues resolved

- WordTemplate created a corrupted file when processing two IF fields
- WordTempalte threw an exception or produced a corrupt file when processing a template with IF fields containing other fields
- Merging a document with a programmatically-created table into another document threw an index out of bounds exception
- After replacing an object in a header or footer using Element.SearchAndReplace, the next call to SearchAndReplace() threw a WordWriterException
- Watermark object can now access watermarks in input files
- Watermark was not able to insert new watermarks
- Setting some table styles using WordApplication caused formatting issues and corrupted tables
- Merging two documents using Document.Append caused corrupted output in some cases
- Opening and saving a file created using a non-English version of Word caused formatting issues
- License Key Manager threw error when being run on a machine with no SoftArtisans license keys in the registry.

Differences between 4.1 and 4.5

Features Added

- WordWriter SharePoint Integration
    - WordWriter can now use SharePoint Lists and Views as data sources
    - WordWriter can now save documents as Document Library items or List Item attachments
- New WordTemplate.SaveExtension property returns the extension of the template file that has been opened
- New Document.Append method appends one Document to another, preserving all Sections, including header and footers plus other section formatting
- WordWriter now supports Grouping and Nesting with the WordTemplate object

Issues resolved

- Improved WordTemplate support for Open Office XML format
    - Newline characters now handled correctly by WordTemplate in the Open Office XML file format
    - Support for all Open Office XML file extensions (.docm, .dotx, and .dotm all are now supported)
- Deleting table rows containing merge fields within a bookmark no longer damages the bookmark, allowing the template to populate without corrupting the output document
- Binding an empty DataTable to a template using WordTemplate.SetRepeatBlock no longer throws an exception, but instead removes bookmark
- Inserting content into a header with field no longer causes a corrupt document
- Custom styles preserved when two Document objects are merged using the Document.InsertAfter method
- Word documents with images with text wrapping settings other than "Inline with Text" no longer corrupted when opened and saved using WordApplication
- Setting TableFormatting.AutoWidth to true no longer fails to have any effect; columns set to Auto using Table.SetPreferredColumnWidthUnits will now have their widths automatically calculated

Differences between 4.0 and 4.1

Features Added

- WordTemplate can evaluate formulas for Word 2007 OOXML file format files (.docx/.docm). For more information, see Creating Merge Fields#formulas

Issues resolved

- Deleting an element in a table made in Word 2007 corrupted the output document in some cases
- Deleting a row in a table also deleted the value in the first cell of the next row
- In some cases documents with embedded images threw Out of Memory exceptions
- Calling Document.Search() on a document with an empty field threw a stack overflow exception
- Inserting many images with WordTemplate caused some images not to display

Quickstart

- Create a Document with WordTemplate
- Create a Document with WordApplication
- Upgrading from WordWriter Version 3 to WordWriter Version 4

Create a Document with WordTemplate

OfficeWriter's WordTemplate object opens a WordWriter template file, populates it with data from one or more data sources, and generates a new Word file. A WordWriter template is a file created in Microsoft Word that contains **merge fields**. A merge field displays a data source field name (for example, a database column name) where a data source value will be inserted. Merge fields are bound to a data source in OfficeWriter code. When you run the code, OfficeWriter replaces the merge fields with values from the data source and creates a new Word file.

Code Sample: Using a Variable as a Data Source

[C#] | [VB.NET]

> WordTemplateCCW.dll, a COM callable wrapper (CCW) that allows you to use the .NET WordTemplate object from ASP, is available upon request. For instructions, see How to Use WordWriter in ASP.

Step 1: Create a Template

To add a merge field to a template, open the Insert menu and select Field. The merge field display format is <<Field Name>>.

To generate a Word file with OfficeWriter, first create a template:

1. In Microsoft Word, create a new file.
2. From the **Insert** menu, select **Field...** to open the Field dialog.
3. From the **Field names** list, select **MergeField**.
4. In the **Field name** text box, enter a name for the merge field. For example, enter *ProductName*.
5. Click Ok. The merge field will be displayed in the document in the format <<Field Name>>.
6. To insert additional merge fields, repeat steps 2-4.
7. Save the template to a folder that OfficeWriter can access at runtime. The ASPNET account or the authenticated user account must have Read access to the template file.

> **The Main Document and Repeat Blocks**
> A WordWriter template may include merge fields in repeat blocks, merge fields in the main document, or both. The "main document" is any part of the template that is not within a repeat block. Merge fields within the main document are bound to a data source by the method SetDataSource. Merge fields within repeat blocks are bound to a data source by the method SetRepeatBlock.

Step 2: Write the OfficeWriter Code

Next, create an ASP.NET script that uses OfficeWriter to open the template, fill in merge field values, and generate a new Word file. The VB.NET and C# samples above generate a file from the template StringVarTemplate.doc. This template contains a single merge field - **<<Product Name>>**. The data source for the merge field is a single-element array.

The WordTemplate object represents the template Word file. The WordTemplate object is in the SoftArtisans.OfficeWriter.WordWriter namespace. The object can be referenced as SoftArtisans.OfficeWriter.WordWriter.WordTemplate. To minimize typing and errors, use an Import directive to import the namespace to the aspx page, and reference the object as WordTemplate, without the namespace prefix:

```
<%@ Import Namespace="SoftArtisans.OfficeWriter.WordWriter" %>
```

To generate a new Word file with OfficeWriter:

1. Create an instance of the WordTemplate object:

C#

```
WordTemplate wt = new WordTemplate();
```

VB.NET

```
Dim wt As New WordTemplate()
```

2. Call WordTemplate.Open to open a template Word file:

C#

```
wt.Open(Server.MapPath("./StringVarTemplate.doc"));
```

VB.NET

```
wt.Open(Server.MapPath("./StringVarTemplate.doc"))
```

The Open method takes the template's path (as in the example), a System.IO.Stream object, a WordApplication object, or a Document object.

> OfficeWriter supports Microsoft Word 97, 2000, 2002 (XP), 2003, and 2007. The default file format for Word 2007, Office Open XML (.docx) is supported by the WordTemplate object. Do not use Open to open files created in earlier versions of Microsoft Word.

3. Call WordTemplate.SetDataSource to bind the template's merge fields to a data source:

C#

```
object[] arrValue = {"SoftArtisans OfficeWriter"};
string[] arrName = {"ProductName"};
wt.SetDataSource(arrValue, arrName);
```

VB.NET

```
Dim arrValue As Object() = {"SoftArtisans OfficeWriter"}
Dim arrName As String() = {"ProductName"}
wt.SetDataSource(arrValue, arrName)
```

This sample application uses a single string variable as a data source. To set the data source to a single value, put the value in a single-element object array, and the merge field name in a single-element string array. Pass the two arrays to WordTemplate.SetDataSource.

4. Call WordTemplate.Process to enter values from the data source into the template's merge fields:

C#

```
wt.Process();
```

VB.NET

```
wt.Process()
```

5. Call WordTemplate.Save to save the generated file or stream it to the browser. In the samples above the file is streamed to the browser:

C#

```
wt.Save(Page.Response, "StringVarOutput.doc", false);
```

VB.NET

```
wt.Save(Page.Response, "StringVarOutput.doc", False)
```

When you pass Save an HttpResponse object (Page.Response) OfficeWriter will stream the file to the browser. The method's second parameter specifies a default name for the generated file; this name will be displayed in the browser's File Download dialog. The third parameter specifies whether the file should open in the browser window or in Microsoft Word; if the parameter is set to False, the file will open in Word.
OfficeWriter also allows you to save the generated file on the server. For more information, see Output Options.

Create a Document with WordApplication

To generate a Word document from code using an API - without using a template - use WordApplication, the main class for pure code-based document generation. This class is an engine used to open, create, and write (save or stream to a browser) documents. A single instance of WordApplication can create multiple documents.

Step 1: Import WordApplication

The WordApplication class is in the SoftArtisans.OfficeWriter.WordWriter namespace. The class can be referenced as SoftArtisans.OfficeWriter.WordWriter. To minimize typing and errors, import the namespace to the aspx page, and reference the class as WordApplication, without the namespace prefix. If you are coding directly in the .aspx page, following the Page directive, include:

```
<%@ Import Namespace="SoftArtisans.OfficeWriter.WordWriter" %>
```

If you are coding in a code-behind page (.aspx.vb or .aspx.cs), include an Imports or using statement at the top of the code-behind page:

C#

```
using SoftArtisans.OfficeWriter.WordWriter;
```

VB.NET

```
Imports SoftArtisans.OfficeWriter.WordWriter
```

Step 2: Create an Instance of WordApplication

To create an instance of WordApplication, use:

C#

```
WordApplication wordApp = new WordApplication();
```

```
Dim wordApp As New WordApplication()
```

Step 3: Create a Document

A Document object represents a Word document. To return a Document object:

- Call WordApplication.Open to open an existing Word file, for example:

C#

```
Document doc = wordApp.Open(@"C:\Reports\Report.doc");
```

VB.NET

```
Dim doc As Document = wordApp.Open("C:\Reports\Report.doc")
```

Or,

- Call WordApplication.Create to create a new Word file, for example:

C#

```
Document doc = wordApp.Create();
```

VB.NET

```
Dim doc As Document = wordApp.Create()
```

Step 4: Add Content to the Document

Each of the following classes represents a distinct section of content in a document. For information on how WordWriter organizes the structure of a Word document, see Word Application Object Tree

Document	A Document object represents the whole Word document.
Section	A Section object represents a major section in a document, like a chapter in a book. Many documents will only contain one section. In more complex documents, content can be divided into multiple sections.
Paragraph	A Paragraph object represents a paragraph in the Word document - a block of text that ends with a carriage return.
CharacterRun	A CharacterRun object represents a contiguous run of characters in the document, all of which have the same formatting.
Table	A Table object represents a table in the document.
TableCell	Each cell in a table is represented by a TableCell object.
List	A List object represents a numbered or bulleted list.
ListEntry	Each entry in a list is represented by a ListEntry object.

To insert a run of characters at the beginning of your document, call Document.InsertTextBefore:

C#

```
CharacterRun charRun =
    doc.InsertTextBefore("This file was created with OfficeWriter", true);
```

VB.NET

```
Dim charRun As CharacterRun = _
    doc.InsertTextBefore("This file was created with OfficeWriter", True)
```

The second parameter of InsertTextBefore specifies whether the inserted text is its own character run with default formatting (true), or if it should assume the formatting of the first character in the Element (in this case, the Document) and become a part of it (false).

You can use the formatting properties of the CharacterRun class to format the text you inserted:

C#

```
charRun.Font.Bold = true;
charRun.Font.FontName = "Times New Roman";
```

VB.NET

```
charRun.Font.Bold = True
charRun.Font.FontName = "Times New Roman"
```

Step 5: Generate a New Word File

When the document is complete, you can save it to the server's hard disk, return it in memory, stream it to the browser, or pass it to WordTemplate (for more information, see Output Options). For example, to stream the document to the browser, use:

C#

```
bool bOpenInBrowser = true;

//--- Open the 'doc' document as "OfficeWriter.doc" in the browser
wordApp.Save(doc, Page.Response, "OfficeWriter.doc", bOpenInBrowser);
```

VB.NET

```
Dim bOpenInBrowser As Boolean = True

'--- Open the 'doc' document as "OfficeWriter.doc" in the browser
wordApp.Save(doc, Page.Response, "OfficeWriter.doc", bOpenInBrowser)
```

When you pass an HttpResponse object to Save, OfficeWriter will stream the generated Word file to the client. The browser will display a download dialog asking the user to open or save the file. The method's third parameter specifies a file name to display in the download dialog. If a user opens the file, and the method's fourth parameter - openInBrowser - is true, the file opens in Internet Explorer. If openInBrowser is false, the file opens in the Word application.

> Word files cannot be embedded in browsers other than IE. For files to appear in a browser, you must enable the "Browse in same window" option. For more information, see Save (Document doc, System.Web.HttpResponse response, String fileName, boolean openInBrowser) and Save (Document doc, System.Web.HttpResponse response, String fileName, boolean openInBrowser, String contentType).

Upgrading from WordWriter Version 3 to WordWriter Version 4

Though WordWriter Version 4 is for the most part backwards compatible with applications written for WordWriter Version 3, there are a number of changes which must be considered when upgrading from Version 3 to Version 4:

- Installing OfficeWriter Version 4
- Changes in WordWriter Properties, Methods, and Enumerations

Installing OfficeWriter Version 4

To upgrade WordWriter Version 3 to WordWriter Version 4, you should first uninstall OfficeWriter Version 3.

Unlike OfficeWriter v3, there are no .NET 1.0 DLLs available for OfficeWriter v4. Before running the OfficeWriter Version 4 installer, you must make sure that you have the .NET runtime 2.0 or higher installed on your server. However, OfficeWriter v4 no longer requires the J# redistributable, so you will not need to install or upgrade J# to install OfficeWriter v4.

To install OfficeWriter v4, follow the installation instructions found in the Installing WordWriter documentation. Remember to use an OfficeWriter v4 or WordWriter v4 license key, as OfficeWriter license keys are version-specific, and an OfficeWriter v3/WordWriter v3 license key will not register correctly with OfficeWriter v4/WordWriter v4.

Changes in WordWriter Properties, Methods, and Enumerations

A number of minor changes were made to WordWriter properties and methods in the shift from J# source code to pure C#.

The following is a list of properties that have been changed to methods in SoftArtisans WordWriter v4. If you use any of these properties in your application, you will need to replace them with the appropriate method calls.

WordWriter v3 Property	WordWriter v4 Method(s)
Document.Bookmark	Document.GetBookmark(String)
DocumentProperties.Property	DocumentProperties.GetProperty(String) DocumentProperties.GetProperty(DocumentProperties.BuiltInProperty) DocumentProperties.SetProperty(String, Object) DocumentProperties.SetProperty(DocumentProperties.BuiltInProperty, Object)
Element.Elements	Element.GetElements(Element.Type)
ParagraphFormatting.Border	ParagraphFormatting.GetBorder(Border.Location)
ParagraphFormatting.Indent	ParagraphFormatting.GetIndent(ParagraphFormatting.IndentLocation) ParagraphFormatting.SetIndent(Int32, ParagraphFormatting.IndentLocation)
Section.Border	Section.GetBorder(Border.Location)
Section.Footer	Section.GetFooter(Section.HeaderFooterType)
Section.Header	Section.GetHeader(Section.HeaderFooterType)
Table.PreferredColumnWidth	Table.GetPreferredColumnWidth(Int32) Table.SetPreferredColumnWidth(Int32, Int32)
Table.PreferredColumnWidthUnits	Table.GetPreferredColumnWidthUnits(Int32) Table.SetPreferredColumnWidthUnits(Int32, TableCell.WidthUnits)
Table.RowHeight	Table.GetRowHeight(Int32) Table.SetRowHeight(Int32, Int32)
Table.RowHeightExact	Table.GetRowHeightExact(Int32) Table.SetRowHeightExact(Int32, Boolean)
TableCell.Border	TableCell.GetBorder(Border.Location)
TableCell.Padding	TableCell.GetPadding(TableCell.Location) TableCell.SetPadding(TableCell.Location, Int32)
TableFormatting.DefaultBorder	TableFormatting.GetDefaultBorder(Border.Location)
TableFormatting.DefaultPadding	TableFormatting.GetDefaultPadding(TableCell.Location) TableFormatting.SetDefaultPadding(TableCell.Location, Int32)

The following is a property that has been changed in SoftArtisans WordWriter v4. If you use this property in your application, you will need to replace it with the new property.

WordWriter v3 Property	WordWriter v4 Property
Position.Position	Position.PositionValue

WordTemplate in Depth

- Creating a Template
- Creating Merge Fields
- Using an Array as a Data Source
- Using a Database as a Data Source
- Importing from Multiple Data Sources
- Importing Multiple Rows
- Creating a Grouped and Nested Document
- WordTemplate Output Options
- Adding Formulas to a Document
- Inserting an Image
- Formatting Merge Fields

Creating a Template

- Creating a Merge Field
- Creating a Repeat Block
- Creating a Mail Merge
- Fast Saved Word Files

- Using an ODBC Data Source to Create Merge Fields
- Example: Creating an Envelope Template
- Example: A Fax Cover Template
- Example: A Form Letter Template

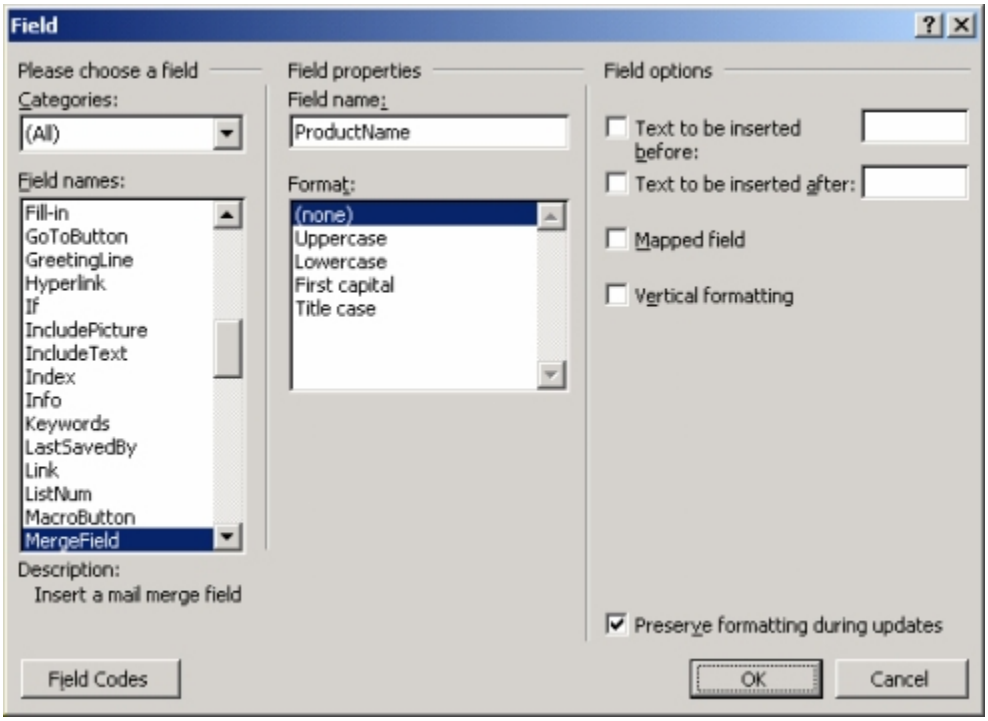
Creating a Merge Field

OfficeWriter's WordTemplate object opens a WordWriter template file, populates it with data from one or more data sources, and generates a new Word file. A WordWriter template is a file created in Microsoft Word that contains **merge fields**. A merge field displays a data source field name (for example, a database column name) where a data source value will be inserted. Merge fields are bound to a data source in OfficeWriter code. When you run the code, OfficeWriter replaces the merge fields with values from the data source and creates a new Word file.

WordTemplate's SetDataSource method binds the merge fields to a data source, which may be an array or a database table record. The Process method enters values from the data source in the template's merge fields.

To create a merge field:

1. In Microsoft Word, open or create a document to use as your WordWriter template.
2. From the **Insert** menu, select **Field...** to open the Field dialog.
3. From the **Field names** list, select **MergeField**.
4. In the **Field name** text box, enter a name for the merge field. For example, enter *ProductName*.
5. Optional: From the **Format** list, select a display format.
6. Optional: In **Field Options**, check **Text to be inserted before** and enter text to insert before the merge field.
7. Optional: In **Field Options**, check **Text to be inserted after** and enter text to insert after the merge field.



The Field Dialog in Microsoft Word 2002 (XP)

Creating a Repeat Block

A repeat block is a text fragment, a list, or a table row in the template document, defined by a Word bookmark, that contains merge fields. In the WordWriter code, a data source is assigned to each repeat block. Repeat blocks allow you to:

- Assign multiple data sources to a single template, and
- Import multiple rows from a data source.

To create a repeat block:

1. In Microsoft Word, open or create a document to use as your WordWriter template.
2. Create a text fragment, a list, or a table row that contains merge fields. For example:

QTY	UNIT ID	DESCRIPTION	UNIT PRICE
«Quantity»	«ProductID»	«ProductName»	«UnitPrice»

3. Select the text, list, or table row that you want to define as a repeat block. For example:

QTY	UNIT ID	DESCRIPTION	UNIT PRICE
«Quantity»	«ProductID»	«ProductName»	«UnitPrice»

4. Open the **Insert** menu and select **Bookmark...** to open the Bookmark dialog.
5. Enter a bookmark name and click **Add**.

In Microsoft Word, bookmarks are not marked by default. To see which document fragments are bookmarks:

1. Open the **Tools** menu and select **Options...**
2. Select the **View** tab.
3. Under **Show**, check **Bookmarks.**

Bookmarks are marked by grey brackets:

QTY	UNIT ID	DESCRIPTION	UNIT PRICE
[«Quantity»	«ProductID»	«ProductName»	«UnitPrice»]

The Main Document and Repeat Blocks
A WordWriter template may include merge fields in repeat blocks, merge fields in the main document, or both. The "main document" is any part of the template that is not within a repeat block. Merge fields within the main document are bound to a data source by the method SetDataSource. Merge fields within repeat blocks are bound to a data source by the method SetRepeatBlock.

Not all Word features can be included in a repeat block:

Supported in Repeat Blocks	Not Supported in Repeat Blocks
• Character Formatting (everywhere) • Paragraph Formatting (Alignment, Outlines, Indent, Spacing, Page Break before, Keep Together) • Multiple Columns (2, 3, Column Breaks) • Borders and Shading • Tabs • Bullets • Numbering (numbering not reset, it is continued from one repeat block to another) • Page Breaks • Section Breaks • Auto Text Field • Hyperlink • Pictures • Table Row Repeat (entire row only) • Fields (with some restrictions)	• Nested repeat blocks (Only one data source may be assigned to a single repeat block.) • Overlapping bookmarks • Comments • Drawing Objects • Text Boxes • Footnote and endnote references • Table and picture indexes • Single cell in a table • Smart tags (smart tags will be removed from the document) • Repeat blocks in headers, footers, footnotes, comments, text boxes, etc.

Creating a Mail Merge

A mail merge is a text fragment, a list, or a table row in the template document that contains merge fields. Unlike repeat blocks, a bookmark is not needed to define the fragment to be repeated. The entire page is treated as a repeat block and repeated as the document is filled with data. The NEXT field can optionally be used to tell WordWriter to move to the next row of data between identical merge fields. Mail merges allow you to import multiple rows from a data source without defining a repeat block.

To create a mail merge:

1. In Microsoft Word, open or create a document to use as your WordWriter template.

2. Create a text fragment, a list, or a table row that contains merge fields. For example:

QTY	UNIT ID	DESCRIPTION	UNIT PRICE
«Quantity»	«ProductID»	«ProductName»	«UnitPrice»

3. Insert Next fields where you would like the data source to move to the next row when it hits a repeat of the same merge fields.

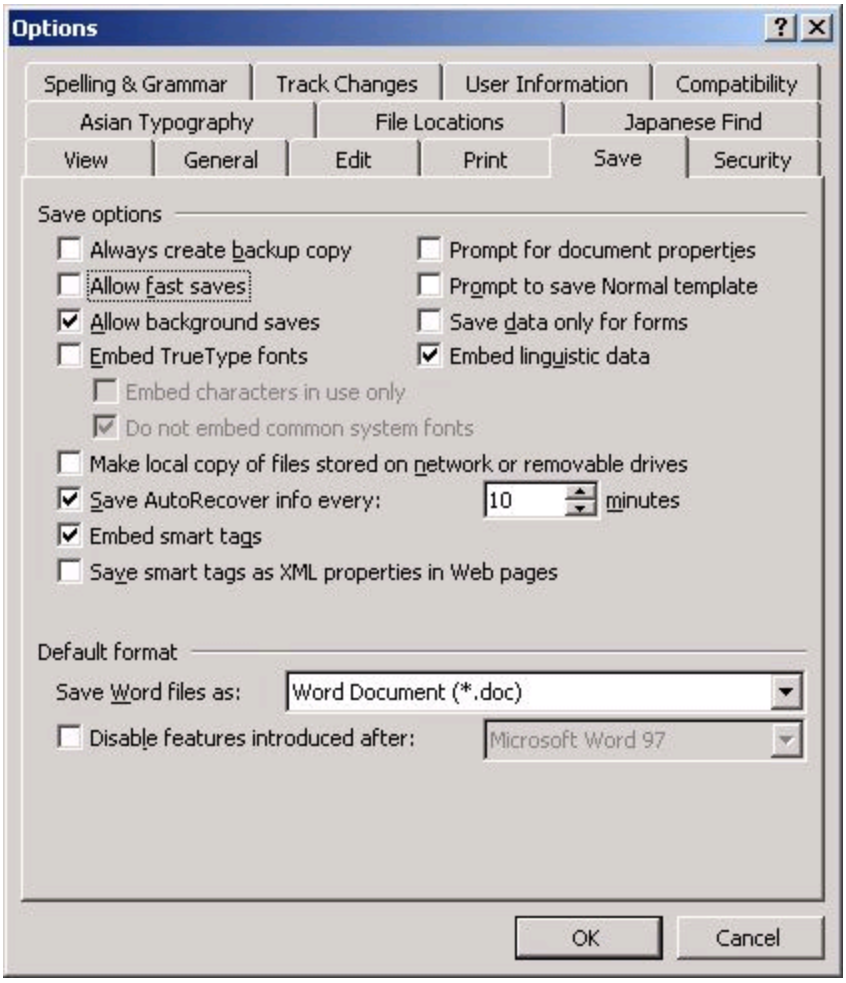
Not all Word features can be included in a Mail Merge:

Supported in Mail Merges	Not Supported in Mail Merges
• Character Formatting (everywhere) • Paragraph Formatting (Alignment, Outlines, Indent, Spacing, Page Break before, Keep Together) • Multiple Columns (2, 3, Column Breaks) • Borders and Shading • Tabs • Bullets • Numbering (numbering not reset, it is continued from one repeat block to another) • Page Breaks • Section Breaks • Auto Text Field • Hyperlink • Pictures • Table Row Repeat • Table Cell Repeat (with Next field) • Fields (with some restrictions)	• Repeat blocks (because a mail merge is treated as a repeat block, adding a repeat block would create a nested repeat block, which is not supported) • Bookmarks • Comments • Drawing Objects • Text Boxes • Footnote and endnote references • Table and picture indexes • Smart tags (smart tags will be removed from the document) • Mail Merges in headers, footers, footnotes, comments, text boxes, etc.

Fast Save Word Files

WordWriter cannot process "fast saved" Word files. Fast saving is a Microsoft Word feature that makes file saving faster by saving only document modifications. When you create a WordWriter template, make sure that fast saves are turned off:

1. In Microsoft Word, open the **Tools** menu.
2. Select **Options...**
3. Select the **Save** tab.
4. Make sure that **Allow fast saves** is not checked.



Using an ODBC Data Source to Create Merge Fields

WordWriter populates merge fields with values from an array or a database. When you use a database as the template's data source, you can use Word's MailMerge toolbar to connect to the database, and select merge fields names from a list of available database fields:

1. In Microsoft Word, open or create a document to use as your WordWriter template.
2. From the **View** menu, select **Toolbars -> Mail Merge**.
3. Click the **Open Data Source** icon.

4. Select a database. For example, open the Access database Northwind.mdb, located in WordWriter\doc-samples\samples\database.
5. Select a database table from the list.
6. Click the **Insert Merge Fields** icon.

7. From the **Fields** list, select a field (a column name) and click **Insert**. Repeat to add additional fields to the template.
8. To format a merge field, right-click the merge field and select **Edit field...** to open the Field dialog. For more information about the Field dialog, see Creating a Merge Field.

Example: Creating an Envelope Template

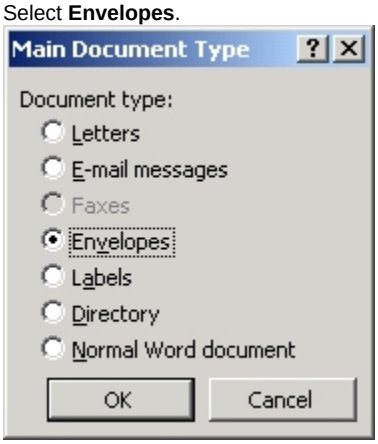
The following VB.NET and C# samples use WordWriter to open an envelope template, set a data source, populate the merge fields, and generate a new envelope. The envelope template contains merge fields for the delivery name and address, and the return name and address. The WordWriter code uses an array as the data source and gets the array values from an HTML form. To run the sample, click Run Sample, enter return and delivery information in the form, and click the Generate button.

Code sample: Using an Envelope Template

[C#] | [VB.NET]

To create an envelope template:

1. Open Microsoft Word.
2. To display the Mail Merge toolbar, open the **View** menu and select **Toolbars -> Mail Merge**.
3. Click the **Main document setup** icon.

4. Select **Envelopes**.


5. In the **Envelope Options** dialog, select envelope size, address fonts, and printing options.
6. Enter merge fields in the envelope's delivery and return address boxes (within the main document). For instructions, see Creating a Merge Field.

Example: A Fax Cover Template

The following VB.NET and C# samples use WordWriter to open a fax cover template, set a data source, populate the merge fields, and generate a new fax cover document. The template contains the merge fields ToName, FromName, FaxNumber, PhoneNumber, Subject, PageCount, Comments, chkUrgent, chkReview, and chkComment. The WordWriter code uses an array as the data source and gets the array values from an HTML form.

Check boxes in the HTML form provide the values for chkUrgent, chkReview, and chkComment. If a box is checked, "X" will be entered in the data source value array and displayed in the generated file. If a box is not checked, an empty string will be entered in the value array and nothing will be displayed by the field name (for example, by "Urgent") in the generated file. To run the sample, click Run Sample, enter the fax information in the form, and click the Generate button.

Code sample: Using a Fax Coversheet Template

[C#] | [VB.NET]
To create a fax cover template using one of Microsoft Word's available templates:

1. In Microsoft Word, open the **File** menu, and select **New...**
2. From the **New Document** window, select **General Templates...**
3. In the **Templates** dialog, select the **Letters & Faxes** tab.
4. Select one of the available fax templates.
5. Insert merge fields in the template.

Example: A Form Letter Template

The following VB.NET and C# samples use WordWriter to open a template form letter, set a data source, populate the merge fields, and generate a new file. The template form letter contains merge fields for recipient name and address and author name and title. The WordWriter code uses an array as the data source and gets the array values from an HTML form. To run the sample, click Run Sample, enter recipient and author information in the form, and click the Generate button.

Code sample: Using a Form Letter Template

[C#] | [VB.NET]

Creating Merge Fields

A WordWriter template is a Microsoft Word file that contains merge fields. A merge field displays a data source field (for example, a database column name). A merge field is created in Microsoft Word and bound in code to a data source. The data source may be an array, a DataSet, a DataTable, or a DataReader. When you run the code, WordWriter populates the merge fields with data source values.

A template may contain multiple sets of merge fields. Each set of fields binds to a single data source. Implicitly or explicitly, a merge field always specifies both a **data source** and a **column**. For example, in the merge field **«Products.ProductID»**, "Products" is the data source and "ProductID" is the column. The merge field **«ProductID»** refers to the "ProductId" column from the first data source defined in the WordWriter code.

Both the data source and the column can be specified by either name or number:

«Products.ProductID»	The "ProductID" column of the data source "Products."
«#2.FirstName»	The "FirstName" column of the second data source defined in the WordWriter code.
«Employee.#3»	The third column of the "Employee" data source.
«#2.#3»	The third column of the second data source defined in the WordWriter code.

Data source and field numbers are 1-based. If WordWriter encounters «#0[.field]» or «[DataSource.]#0», an error will occur.

The default data source for a main document merge field is the first data source set in the WordWriter code (by the method SetDataSource). The default data source for a merge field within a repeat block is the data source set for the repeat block (by the method SetRepeatBlock, which specifies the repeat block by bookmark name). You can include main document merge fields within a repeat block, however, these fields will not be repeated when the template is populated (only one row of data can be imported to main document merge fields). The data source for a repeat block is only valid within the repeat block; you cannot include a repeat block merge field outside of its bookmark-defined repeat block.

The Main Document and Repeat Blocks
A repeat block is a text fragment, a list, or a table row - defined by a Word bookmark - that contains merge fields and that will be repeated for each row returned from the database. The main document is any part of the template that is not within a repeat block. You can assign one data source to each repeat block, and multiple data sources to the main document. Only one row of data can be imported to each set on main document merge fields.

- Merge Fields Names
- Using Short Merge Fields
- Using Long Merge Fields
- Merge Field Modifiers
- Merge Field Formulas

Merge Field Names

Follow these rules when naming merge fields:

- Data source and column names must not include Unicode characters.
- Data source and column names must begin with a letter (A-Z, a-z).
- Data source and column names may include the following characters only:
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890_

  If you assigned a different data source separator, you can also use a "." in your data source and column names.

- Spaces are not allowed anywhere in a merge field.

Use brackets if your column name does not conform to these rules
The exception of these rules is when a column name is enclosed in brackets (for example, «Products.[Product Category]» . To include spaces or unicode characters in the data source column name, use this format:«DataSource.[Column Name]»
For example: «Employees.[Street Address]» . The [Column Name] format allows you to match any SQL column name exactly. Legal characters within the brackets include every character except parentheses () or brackets [].

Using Short Merge Fields

WordWriter supports several full-length and short merge field forms. When you use any of the long merge field formats, you explicitly include both the data source name or number and the column name or number. Short formats do not specify both data source and column explicitly; one or the other is omitted. How WordWriter understands a short merge field depends on whether or not the marker contains a $.

WordWriter understands «$Employee» as the first column of the data source "Employee." If you omit the $, WordWriter will read "Employee" as a column from the first data source defined in the WordWriter code.

- «$Employee» means the first column in the data source "Employee."
- «Employee» means the "Employee" column in the first data source defined in the WordWriter code.

Examples

Form	Example
«#ColumnNumber»	«#7» Seventh column of the first data source defined in the WordWriter code.
«$#DataSourceNumber»	«$#7» First column of the seventh data source defined in the WordWriter code.
«ColumnName»	«OrderId» "OrderId" column from the first data source defined in the WordWriter code.
«$DataSourceName»	«$Orders» First column in the data source "Orders."

If you are using the WordTemplate.SetMailMerge method, you **must** use Short Merge fields.

Using Long Merge Fields

When you use any of the long merge field formats, you explicitly include both the data source name or number and the column name or number. Using the long formats may prevent confusion when writing the WordWriter code, since you must be able to distinguish between data source and column to set a data source in code.

Examples

Form	Example
«DataSourceName.ColumnName»	«Employees.FirstName» "FirstName" column in the "Employees" data source.
«DataSourceName.#ColumnNumber»	«Products.#3» Third column of the data source "Products."
«#DataSourceNumber.ColumnName»	«#2.ProductId» "ProductId" column of the second data source defined in the WordWriter code.
«#DataSourceNumber.#ColumnNumber»	«#3.#2» Second column of the third data source defined in the WordWriter code.
«DataSourceName.[Column Name]»	«Employees.[Street Address]» "Street Address" column of the data source "Employees."

Merge Field Modifiers

An WordWriter merge field can include modifiers. Modifiers are included in parenthesis at the end of the merge field. For example:

```
&laquo;Products.ProductID(Lowercase)&raquo;
```

If a merge field includes more than one modifier, the modifiers should be separated by commas. Do not include spaces between modifiers.

WordWriter supports the following merge field modifiers:

Fieldname	Use the Fieldname modifier to insert database column headings in a document. For example, in place of the merge field «RecordSet.#2(fieldname)», WordWriter will insert the name of the second column in the RecordSet.
Uppercase	If a merge field includes the Uppercase modifier, all text values imported to the merge field will be displayed in uppercase. If a merge field includes both the Uppercase and Lowercase modifiers, the last will be applied.
Lowercase	If a merge field includes the Lowercase modifier, all text values imported to the merge field will be displayed in lowercase. If a merge field includes both the Uppercase and Lowercase modifiers, the last will be applied.

Optional	By default, if a merge field's data source name or column name is not bound to a data source in code, an error will occur. If a merge field contains the Optional modifier, and the merge field is not bound by name to a data source, WordWriter will discard the merge field in the generated document and will not throw an error. This modifier makes both a merge field's data source and column optional. For example, for this merge field: «Orders.OrderID(Optional)» If either the data source "Orders," or the column "OrderId" does not exist, the merge field will be removed without error. If a merge field contains a data source number and/or field number (for example «#7.#3») and is not bound to a data source in script, the merge field will be removed without error whether the "Optional" modifier is present or not.

Merge Field Formulas

Word formulas are only supported with the Word 2007 OOXML file format (.docx/.docm).

Beginning with WordWriter 4.1, the WordTemplate object evaluates a number of different formulas placed in merge fields using the format «=FORMULA(DataSource.Field)». Each of the formulas will perform a specific operation on the values of the referenced data field. Formulas are evaluated for all values of the field in the data source, regardless of how many of the values are actually displayed in the document.

WordWriter supports the following merge field formulas:

AVERAGE	Calculates the average of the field values
COUNT	Returns the number of values (that is, the number of rows in the data source) of the field
COUNTA	Returns the number of non-null values of the field
MAX	Returns the maximum value from the field
MIN	Returns the minimum value from the field
PRODUCT	Calculates the product of the field values
STDEV	Returns the standard deviation of the field values, treating the values as a sample
STDEVP	Returns the standard deviation of the field values, treating the values as a population
SUM	Calculates the sum of the field values
VAR	Returns the variance of the field values, treating the values as a sample
VARP	Returns the variance of the field values, treating the values as a population

Using an Array as a Data Source

The following example generates a Word file from a template that contains the merge fields ProductName, CompanyName, and URL. The WordWriter code gets the values for these fields from an array of objects.

To set a template's data source to an array, first create an object array of field values and a string array of field names:

C#

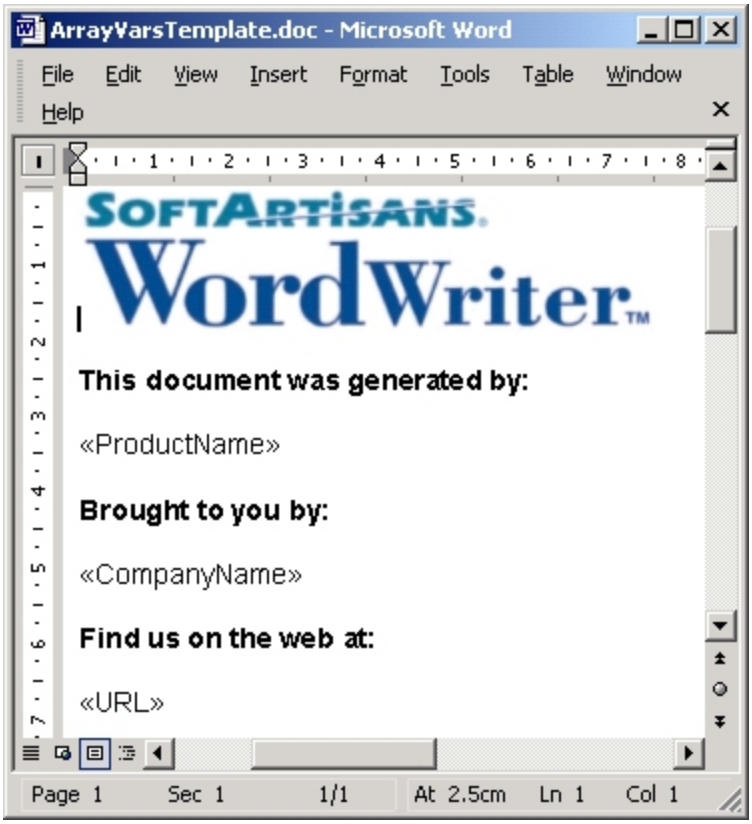
```
//--- Create an object array of field values
object[] arrValue = {"SoftArtisans", "WordWriter", "http://www.softartisans.com"};

//--- Create a string array of field names
string[] arrName = {"CompanyName", "ProductName", "URL"};
```

VB.NET

```
'--- Create an object array of field values.
Dim arrValue As Object() = {"SoftArtisans", "WordWriter", "http://www.softartisans.com"}

'--- Create a string array of field names.
Dim arrName As String() = {"CompanyName", "ProductName", "URL"}
```



Each merge field in the template will bind to a field name/field value pair. When creating field name and field value arrays to use as a data source, remember:

- The two arrays must contain the same number of elements.
- Each name in the array of field names must be the same as the corresponding merge field name in the template.
- The number of unique merge fields in the template may not exceed the number of values in the data source. The data source must provide a value for each merge field in the template.
- The number of values in the data source may be greater than the number of merge fields in the template.

First, create the WordTemplate object and open a template file:

C#

```
string templatePath;
 WordTemplate wt = new WordTemplate();
 wt.Open(templatePath);
```

VB.NET

```
Dim templatePath As String
 Dim wt As New WordTemplate()
 wt.Open(templatePath)
```

You can call the SetDataSourcemethod before or after opening the template (that is, before or after calling Open). When you call SetDataSource,

pass the two arrays you created to the method as parameters:

C#

```
wt.SetDataSource(arrValue, arrName);
```

VB.NET

```
wt.SetDataSource(arrValue, arrName)
```

The Process method enters the array values in the template's merge fields:

C#

```
wt.Process();
```

VB.NET

```
wt.Process()
```

The Save either saves the new Word file on the server, or streams it to the browser. In this example, WordWriter streams the generated file to the client:

C#

```
wt.Save(Page.Response, "ArrayVarsOutput.doc", false);
```

VB.NET

```
wt.Save(Page.Response, "ArrayVarsOutput.doc", False)
```

On the client, a File Download dialog will ask the user to open or save the Word file. Since the Save method's third parameter - OpenInBrowser - is set to False, if the user chooses to open the file, it will open in Microsoft Word.

Using a Database as a Data Source

You can use a database table as a data source for a WordWriter template by passing the SetDataSource method a DataTable, a DataSet, an SqlDataReader, OleDbDataReader or an AdomdDataReader. If the specified data source contains more than one row, SetDataSource will use the first row of the first table as the data source. To import multiple rows from data sources use a repeat block.

Import the System.Data namespace for DataTable and DataSet objects. Import System.Data.SqlClient for SQL Server-specific database classes, or System.Data.OleDb for the OLEDB client and connections to other databases such as Microsoft Access .MDB files.

The following samples use SqlClient classes and connect to the AdventureWorks2000 database. These code samples use ADO.NET objects as data sources:

Code Samples

Code Sample: Purchase Order Document

[C#] | [VB.NET]

Code Sample: Mailing Labels

[C#] | [VB.NET]

Code Sample: Mail Merge

[C#] | [VB.NET]

Using an ADO.NET DataTable or DataSet as a Data Source

The following code example shows how to query the AdventureWorks2000 database and return the data as a DataSet object. The DataSet object is then used as the data source in a WordTemplate document:

C#

```
void GenerateDocument(int employeeID)
{
     //--- Query the database
     DataTable dt = new DataTable();
     using(SqlConnection conn = new SqlConnection(connString))
     {
          string sql =
               "SELECT FirstName, LastName FROM Employee WHERE EmployeeId = @id";
          SqlCommand cmd =
               new SqlCommand(sql, conn);

          //--- Use the employee ID as a SQL parameter
          cmd.Parameters.Add("@id", employeeID);
          SqlDataAdapter adpt = new SqlDataAdapter(cmd);
          adpt.Fill(dt);
     }

     //--- Use a DataTable as the data source
     WordTemplate wt = new WordTemplate();
     wt.Open(templatePath);
     wt.SetDataSource(dt);
     wt.Process();
     wt.Save(Page.Response, "DatabaseOutput.doc", false);
}
```

VB.NET

```
Private Sub GenerateDocument(ByVal employeeID As Integer)
     '--- Query the database
     Dim dt As New DataTable()
     Dim conn As New SqlConnection(connString)
     Dim sql As String = _
          "SELECT FirstName, LastName FROM Employee WHERE EmployeeId = @id"
     Try
          Dim cmd As New SqlCommand(sql, conn)

          '--- Use the employee ID as a SQL parameter
          cmd.Parameters.Add("@id", employeeID)
          Dim adpt As New SqlDataAdaptercmd)
          adpt.Fill(dt)
     Finally
          If Not conn Is Nothing Then
               conn.Dispose()
          End If
     End Try

     '--- Use a DataTable as the data source
     Dim wt As New WordTemplate()
     wt.Open(templatePath)
     wt.SetDataSource(dt)
     wt.Process()
     wt.Save(Page.Response, "DatabaseOutput.doc", False)
End Sub
```

The first block of code in the GenerateDocument method connects to a database using a variable *connString* which represents a SQL Server connection string. Then, a SqlCommand is created and a parameter is added to the query. A SqlDataAdapter then executes the SqlCommand and fills a DataTable with the data. The next block of code shows how to use this DataTable with WordTemplate.

You can also use a DataSet as the data source. When using a DataSet, WordTemplate will use only the first DataTable in the DataSet.Tables collection.

Using an SqlDataReader, OleDbDataReader or AdomdDataReader as a Data Source

SqlDataReader, OleDbDataReaders and AdomdDataReader are also available for use by WordTemplate. Remember that unlike DataSet and DataTable objects, DataReader objects require an open connection to the database while reading data. Remember to close SqlDataReader, OleDbDataReader or AdomdDataReader objects calling WordTemplate.Process().

C#

```
SqlCommand cmd;

//--- CloseConnection will close the associated connection
//--- when the reader is closed
SqlDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);

try
{
     WordTempalte wt = new WordTemplate();
     wt.Open(templatePath);
     wt.SetDataSource(rdr);
     wt.Process();
     wt.Save(Page.Response, "DataReader.doc", false);
}
finally
{
     if(rdr!=null)
          rdr.Close();
}
```

VB.NET

```
Dim cmd As SqlCommand

'--- CloseConnection will close the associated connection when
'--- the reader is closed
Dim rdr As SqlDataReader = cmd.ExecuteReader(CommandBehavior.CloseConnection)

Try
     Dim wt As New WordTemplate();
     wt.Open(templatePath)
     wt.SetDataSource(rdr)
     wt.Process()
     wt.Save(Page.Response, "DataReader.doc", False)
Finally
     If Not rdr Is Nothing Then
          rdr.Close()
     End If
End Try
```

Importing from Multiple Data Sources

In a WordWriter template, a merge field displays a data source field (for example, a database column name) where a data source value will be inserted. A template may include merge fields in repeat blocks, merge fields in the main document, or both. A repeat block is a text fragment, a list, or a table row in the template document, defined by a Word bookmark, that contains merge fields. The main document is any part of the template that is not within a repeat block.

WordWriter provides two methods for binding merge fields to a data source: SetDataSource and SetRepeatBlock. SetDataSource sets a data source for merge fields in the main document. SetRepeatBlock sets data sources for merge fields within repeat blocks.

The main document may include multiples sets of merge fields, where each set of fields binds to a different data source. A SetDataSource call is required for each set of main document merge fields. Only one row of data can be imported to each set of main document merge fields. When you include more than one set of merge fields in the main document, use any of the following SetDataSource methods to set data sources:

C#

```
SetDataSource(object[] data, string[] fieldNames, string name)
SetDataSource(DataSet ds, string name)
SetDataSource(DataTable dt, string name)
SetDataSource(IDataReader dr, string name)
```

VB.NET

```
SetDataSource(data() As Object, fieldNames() As String, name As String)
SetDataSource(ds As DataSet, name As String)
SetDataSource(dt As DataTable, name As String)
SetDataSource(dt As IDataReader, name As String)
```

The SetDataSource method is described in fuller detail on this page in our product documentation: SetDataSource.

A template can include multiple repeat blocks. Each repeat block binds to a different data source. A SetRepeatBlock call is required for each repeat block in the template. A repeat block will be repeated for each row returned from the data source.

The "Purchase Order Document" code sample generates a Word file from a template that contains a set of merge fields in the main document, and a set of merge fields within a single repeat block. The repeat block's bookmark name is "DetailRow." The code assigns two data sources to the template. setDataSource binds an array data source to the main document's merge fields, and setRepeatBlock binds a ResultSet data source to the merge fields within the "Detail Row" repeat block.

Code Samples

Code Sample: Purchase Order Document

[C#] | [VB.NET]

Code Sample: Using a Variable as a Data Source

[C#] | [VB.NET]

Importing Multiple Rows

Use a **repeat block** to import multiple rows from a data source. A repeat block is a fragment in the template document - defined by a Word bookmark - that contains merge fields and that will be repeated for each row in a data source. To import multiple rows from a single data source, create a repeat block in the template and, in the WordWriter code, call SetRepeatBlock to bind the repeat block to a data source.
The main document is any part of the template that is not within a repeat block. SetDataSource sets a **single-row** data source for merge fields in the main document. If you pass SetDataSource a ResultSet that contains more than one row, WordWriter will import the first row to the template. Multiple row data sources can be imported to repeat blocks only, not to main document merge fields.

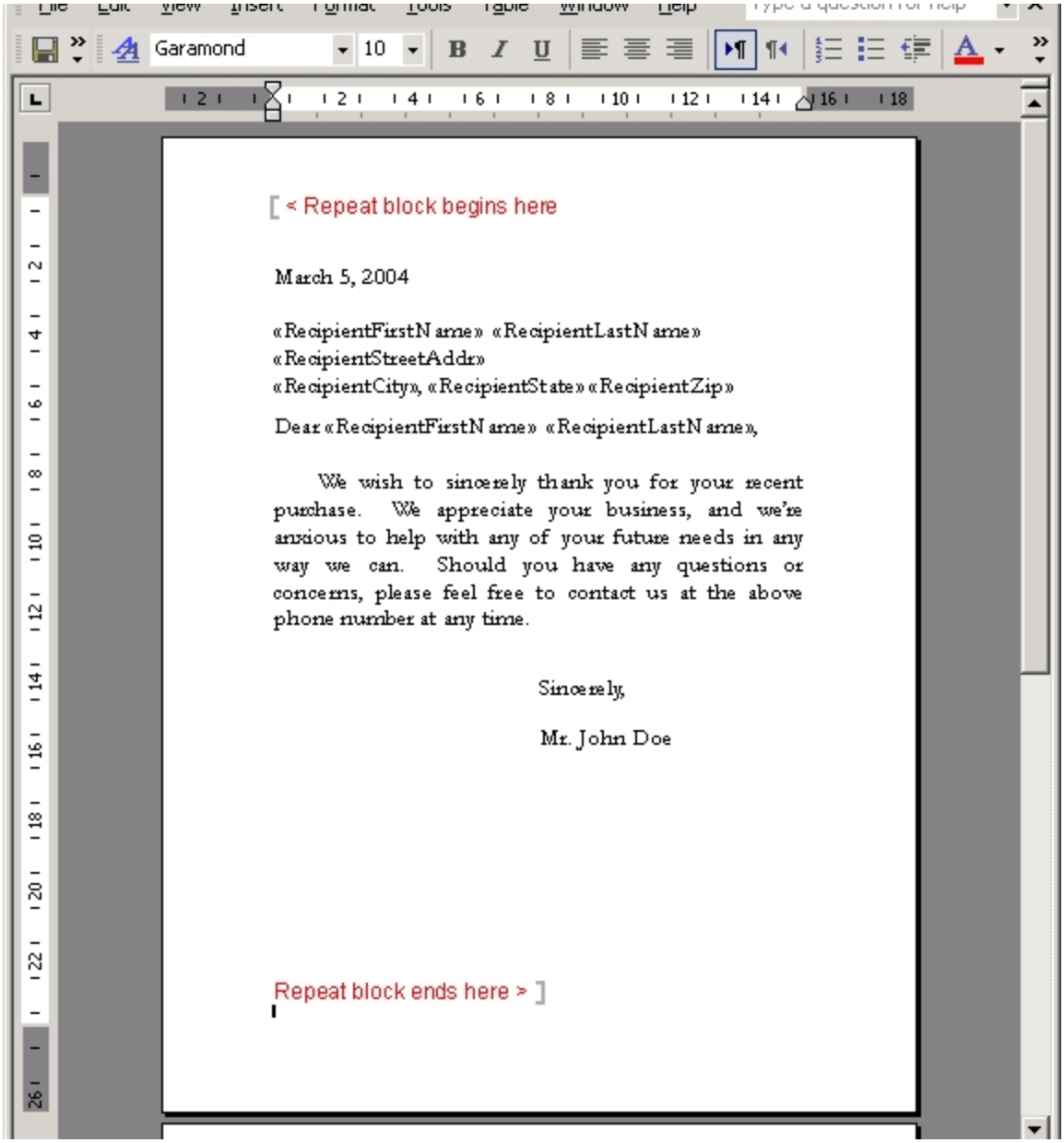
To create a repeat block:

1. In Microsoft Word, create a text fragment, a list, or a table row that contains merge fields.
2. Select the text, list, or table row that you want to define as a repeat block.
3. Open the **Insert** menu and select **Bookmark...** to open the Bookmark dialog.
4. Enter a bookmark name and click **Add**.

In Microsoft Word, bookmarks are not marked by default. To see which document fragments are bookmarks:

1. Open the **Tools** menu and select **Options...**
2. Select the **View** tab.
3. Under **Show**, check **Bookmarks**. Bookmarks will be marked by grey brackets.

Whether the template will bind to one data source or several, merge fields are not required in the main document. To import multiple rows from a single data source, use a repeat block only, and do not include any merge fields in the main document. The repeat block can span all the content in the document. For example, by defining a repeat block with a bookmark that spans an entire page, and setting a multiple-row data source, you can use WordWriter to create multiple form letters:



Here is some example code that demonstrates the usage of SetRepeatBlock with a DataTable that has multiple rows. This code inserts data into merge fields "FirstName" and "LastName" enclosed in the repeat block named "BookMarkName".

C#

```
void GenerateDocument(int employeeID)
{
     //--- Query the database
     DataTable dt = new DataTable();
     using(SqlConnection conn = new SqlConnection(connString))
     {
          string sql =
               "SELECT FirstName, LastName FROM Employee";
          SqlDataAdapter adpt = new SqlDataAdapter(sql, conn);
          adpt.Fill(dt);
     }

     //--- Use a DataTable as the data source
     WordTemplate wt = new WordTemplate();
     wt.Open(templatePath);
     wt.SetRepeatBlock(dt, "BookMarkName");
     wt.Process();
     wt.Save(Page.Response, "RepeatBlockOutput.doc", false);
}
```

VB.NET

```
Private Sub GenerateDocument(ByVal employeeID As Integer)
     '--- Query the database
     Dim dt As New DataTable()
     Dim conn As New SqlConnection(connString)
     Dim sql As String = _
          "SELECT FirstName, LastName FROM Employee""
     Try
          Dim adpt As New SqlDataAdapter(sql, conn)
          adpt.Fill(dt)
     Finally
          If Not conn Is Nothing Then
               conn.Dispose()
          End If
     End Try

     '--- Use a DataTable as the data source
     Dim wt As New WordTemplate()
     wt.Open(templatePath)
     wt.SetRepeatBlock(dt, "BookMarkName")
     wt.Process()
     wt.Save(Page.Response, "RepeatBlockOutput.doc", False)
End Sub
```

Code Sample: Mail Merge using Repeat Blocks

The following code sample demonstrates the use of SetRepeatBlock to emulate mail merge behavior. The repeat block spans an entire page within the document. This causes a new page to be created for each recipient in the database:

[C# | VB.NET]

Creating a Grouped and Nested Document

Introduced in build 4.5

> ⚠ Grouping and Nesting is only available in WordWriter EE.

Beginning in version 4.5, the WordTemplate object includes support for nested bookmarks with Office Open XML documents (.docx, .docm, .dotx, and .dotm) in WordWriter Enterprise Edition. Nested bookmarks can be used to format and display hierarchical data.

The Grouping Repeat Block

Grouping and Nesting with the WordTemplate object is a modification of existing Repeat Block and Mail Merge functionality used to import multiple rows. Standard repeat blocks and mail merges are not able to display data hierarchically because they cannot be nested. Grouping blocks resolve this problem by allowing you to nest a grouping block within a repeat block or a mail merge document, controlling which data will be repeated for each primary value.

A Grouping block is denoted by a bookmark with a name of the format "group_fieldname" if you use field names in your merge fields or "groupcolumn_1" if you use ordinal values in your merge fields.

Below is an example of how a repeat block A would look with a grouping block B within it. The repeat block A is a bookmark that can have any name, for example "Repeat". The grouping block B is a bookmark within the "Repeat" bookmark, which has the name "group_primary". When SetRepeatBlock is called on "Repeat", it will be grouped by the field «primary». The header and footer, here colored red, will be repeated for every different value in the field, while the grouping block, here colored black, will be repeated for every row in the data source.

A: [Header «primary»
B: [«first» «second»]
footer]

If you have multiple grouping blocks that group by the same field, you can use a number before the underscore, for example "group1_fieldname" and "group2_fieldname".

Bookmark names are limited in Microsoft Word to fewer than 40 characters and cannot contain special characters – that is, bookmarks can only contain letters, numbers or underscores. If your grouping field name is too long or has ampersands or other special characters, you can use a grouping alias merge field modifier to give your fields nicknames for grouping purposes. For instance, if you want to group by a field called «[Bob&Alice]», you can add a modifier with a new name to be used in grouping: «[Bob&Alice](groupAlias(alias))». You can then group by that column by using a bookmark with the name "group_alias".

How to Group using the Grouping Block

The Grouping block is used to separate headers and footers, which are only repeated for every primary field value, from the parts of the document that are repeated for each row. The header is the section within the repeat block or mail merge before the grouping repeat block, which will be repeated for every value in the primary (grouping) field. The footer is the section within the repeat block or mail merge after the repeat block, which will also be repeated only for each new value in the primary field. Everything within the grouping block, then, is repeated for each new row in the data source. For example, use the following data source:

Name	Purchase	Quantity
Frank	Football	3
Frank	Basketball	3
Frank	Athletic Shorts	2
Frank	Athletic Shirt	2
Frank	Football Cleats	1
Frank	Basketball Hi-Tops	1
Stephanie	Basketball	2
Stephanie	Athletic Shorts	2
Stephanie	Athletic Shirt	1
Stephanie	Basketball Hi-Tops	1
Ian	Football	3
Ian	Basketball	3
Ian	Athletic Shorts	2
Ian	Athletic Shirt	2
Ian	Football Cleats	1
Ian	Basketball Hi-Tops	1

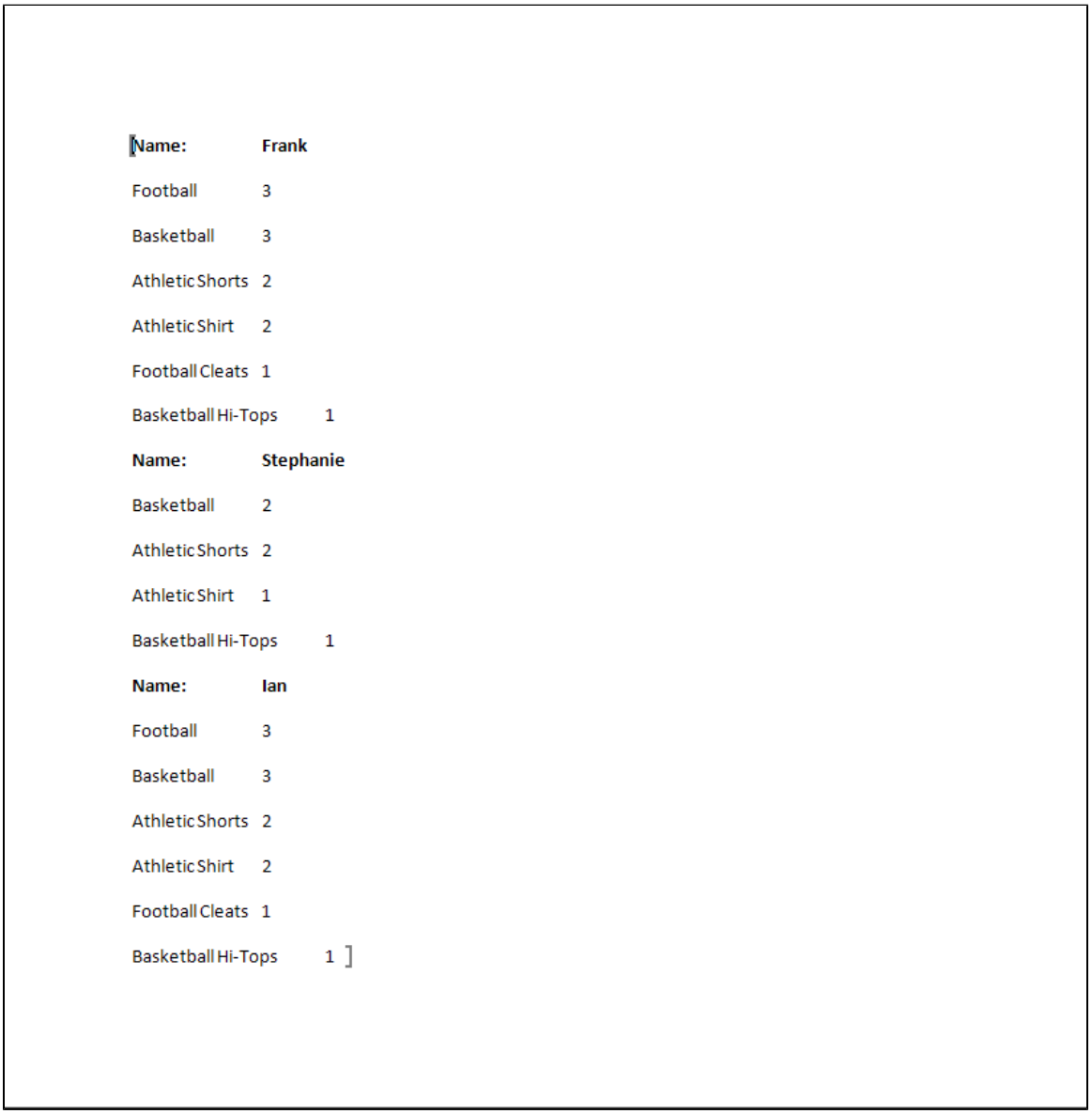
The following grouping block will group on the first column, including a header for each different value of the first column:

Note that you can only see three bookmark indicators ([or]). In Word, if two bookmarks end at the same place, only a single bookmark closed indicator will be shown.

With the outer bookmark named "Repeat" and the inner bookmark named "Groupcolumn_1", the following code can be used to bind data to the grouping block:

```
wt.SetRepeatBlock("repeat", data);
```

The resulting document will look like this:



Grouping with Mail Merges

Hierarchical data can be grouped and nested within Mail Merges as well as Repeat Blocks. The above sample could be created with a Mail Merge by simply deleting the "repeat" bookmark and using the following code to bind data:

```
wt.SetMailMerge(data);
```

The resulting document will be the same, as SetMailMerge will treat the content of the page as if it were a repeat block.

Grouping with the NEXT field

Additionally, NEXT fields are evaluated sequentially within and between grouping blocks. A NEXT field tells WordWriter to move to the next row of the data source, which can be helpful if you wish to use alternating styles using either SetMailMerge or SetRepeatBlock. If you want to alternate styles within a group, place a NEXT field within the grouping block. If you want to alternate styles for each value in the grouped column, you will need to use multiple groups (with names such as group1_Field and group2_Field) with a NEXT block between them.

For example, if I want to add a little color to the previous example, I can add another row to the grouping block, separated by a NEXT field, that is colored blue. If I want to add even more color, I can add another grouping block in red, separated from the first by a NEXT field. The template is shown below with the "Show codes" feature activated so that you can see the NEXT fields:

```
        {MERGEFIELD Name \b "Name: " \* MERGEFORMAT}
group1_Name
        [{MERGEFIELD Purchase \* MERGEFORMAT}  { MERGEFIELD Quantity \* MERGEFORMAT}{ NEXT
        \* MERGEFORMAT}

        {MERGEFIELD Purchase \* MERGEFORMAT}  {MERGEFIELD Quantity \* MERGEFORMAT}]

        {NEXT \* MERGEFORMAT}

        {MERGEFIELD Name \b "Name: " \* MERGEFORMAT}
group2_Name
        [{MERGEFIELD Purchase \* MERGEFORMAT}  { MERGEFIELD Quantity \* MERGEFORMAT}]
```

The output file will look like this:

Name: Frank

Football 3

Basketball 3

Athletic Shorts 2

Athletic Shirt 2

Football Cleats 1

Basketball Hi-Tops 1

Name: Stephanie

Basketball 2

Athletic Shorts 2

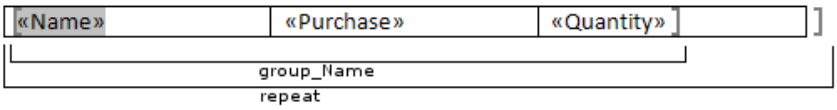
Athletic Shirt 1

Basketball Hi-Tops 1

Grouping in Tables

One common use of Grouping and Nesting functionality is in tables in a Word document. Grouping and Nesting functionality can be used to create a variety of different tables, but two examples are featured below: creating a single table that removes repeated values and creating a new table for each value in a column.

The first example uses the data set above with a template that has a single grouping bookmark, "group_Name", on a row in a table, with the repeat block "repeat" covering the entire row. There is an important distinction to how you create a group on a row, as opposed to a repeat block on a row: to create the group on the row, select all the cells of the row and create the bookmark; to create the repeat block on the row, select the row itself and create the bookmark. Below is a picture of the template, which has a grouping block on all the cells of the row, which ends within the last cell, and a repeat block over the entire row, which ends outside of the table boundary:



When processed using WordTemplate's SetRepeatBlock method, the follow document results:

Frank	Football	3
	Basketball	3
	Athletic Shorts	2
	Athletic Shirt	2
	Football Cleats	1
	Basketball Hi-Tops	1
Stephanie	Basketball	2
	Athletic Shorts	2
	Athletic Shirt	1
	Basketball Hi-Tops	1
Ian	Football	3
	Basketball	3
	Athletic Shorts	2
	Athletic Shirt	2
	Football Cleats	1
	Basketball Hi-Tops	1

The next sample uses data derived from the AdventureWorks database. For more details, see the Nested Mail Merge sample.

The following template is used to compose a letter to customers with a new table for each new purchase. The header includes the logo, an address to the customer, and a paragraph explaining the summary of purchases. There are actually two grouping repeat blocks. The first, "group_CustomerId", includes the entire table and the label, "Purchased on «orderdate»", and will be repeated for each value of the CustomerId field. The second, "group_SalesOrderId", includes the inner content of the table and will be repeated for each value of the SalesOrderId field. The outer repeat block, "repeat", includes the entire page.

Dear «FirstName» «LastName»,

This is your annual summary of purchases made in the last year from the AdventureWorks bicycle company. Please review the statement and contact us to discuss any errors or inaccuracies.

Purchased on «orderdate»

Item	Quantity	Unit Price	Line Total
«name»	«orderqty»	«unitprice»	«linetotal»
		Subtotal	**«subtotal»**
		Tax	**«taxamt»**
		Total Cost	**«totaldue»**

]

]

After binding the template with data, the following document is the result:

Dear Jon Yang,

This is your annual summary of purchases made in the last year from the AdventureWorks bicycle company. Please review the statement and contact us to discuss any errors or inaccuracies.

Purchased on July 22, 2001

Item	Quantity	Unit Price	Line Total
Mountain-100 Silver, 38	1	$3399.99	$3399.99
		Subtotal	**$3399.99**
		Tax	**$272.00**
		Total Cost	**$3756.99**

Purchased on July 22, 2003

Item	Quantity	Unit Price	Line Total
Mountain-200 Silver, 38	1	$2319.99	$2319.99
Fender Set - Mountain	1	$21.98	$21.98
		Subtotal	**$2341.97**
		Tax	**$187.36**
		Total Cost	**$2587.88**

Purchased on November 04, 2003

Item	Quantity	Unit Price	Line Total
Touring-1000 Blue, 46	1	$2384.07	$2384.07
Touring Tire	1	$28.99	$28.99
Touring Tire Tube	1	$4.99	$4.99
Sport-100 Helmet, Red	1	$34.99	$34.99
Short-Sleeve Classic Jersey, S	1	$53.99	$53.99
		Subtotal	**$2507.03**
		Tax	**$200.56**
		Total Cost	**$2770.27**

Of course, further pages will continue the orders, and a new letter is composed for each customer.

WordTemplate Output Options

After populating your WordWriter template from a data source, call Save to either save the new Word file on the server, or stream it to the browser. WordWriter can:

- Save the generated Word file to disk .
- Stream the Word file to the client and open it in the browser window .
- Stream the Word file to the client and open it in the default application for .doc and .docx Files .
- Save the generated Word file on the server and stream it to the client .
- Pass the template to WordApplication .

> When WordWriter streams a file to the client, the output result will depend on the browser as well as the server-side script. For example, when WordWriter streams a file directly to the browser or to Microsoft Word, the browser may append a number to the name of the file (because it caches a copy of the file and files must have unique names). The best way to control download behavior on the client is use OfficeWriter Assistant, which is available with WordWriterEE.

Save to Disk

To save the generated file on the server:

Pass the Save method a file path, for example:

C#

```
WordTemplate wt = new WordTemplate();
wt.Process();
wt.Save(@"c:\Documents\File.doc");
```

VB.NET

```
Dim wt As New WordTemplate()
wt.Process()
wt.Save("c:\Documents\File.doc")
```

OR

Pass the Save method a System.IO.FileStream object, for example:

C#

```
using System.IO; // For FileStream object
WordTemplate wt = new WordTemplate();
wt.Process();
using(FileStream fstream = new FileStream(@"c:\Documents\File.doc",
  FileMode.Create))
{
 wt.Save(fstream);
 fstream.Close();
}
```

VB.NET

```
Imports System.IO ' For FileStream object
Dim wt As New WordTemplate()
wt.Process()
Dim fstream As new FileStream("c:\Documents\File.doc",
  FileMode.Create)
Try
 wt.Save(fstream)
 fstream.Close()
Finally
 If Not fstream Is Nothing Then
  fstream.Dispose()
 End If
End Try
```

Open in the Browser

When you pass an HttpResponse object to Save, WordWriter will stream the generated Word file to the client. The browser will display a File Download dialog asking the user to open or save the file. The method's second parameter specifies a file name to display in the File Download dialog. If the method's third parameter - OpenInBrowser - is True and the user chooses to open the Word file, the file will open in the browser window.

C#

```
wt.Save(Page.Response, "StringVarOutput.doc", true);
```

VB.NET

```
wt.Save(Page.Response, "StringVarOutput.doc", True)
```

Alternatively, stream the file to the browser by passing Response.Stream to the Save method. In this case, to open the generated file in the browser window:

1. Set the response's content-type header to "application/vnd.ms-word":

```
Response.ContentType = "application/vnd.ms-word"
```

2. Set the response's content-disposition to "inline":

```
Response.AddHeader("Content-Disposition", "inline;filename=""WordWriter.doc""")
```

3. Pass the Save method Response.Stream as a parameter:

```
oWW.Save(Response.OutputStream)
```

Open in the Default Application for .doc or .docx Files

When you pass an HttpResponse object to Save, WordWriter will stream the generated Word file to the client. The browser will display a File Download dialog asking the user to open or save the file. The method's second parameter specifies a file name to display in the File Download dialog. If the method's third parameter - OpenInBrowser - is False and the user chooses to open the Word file, the file will open in the default application for .doc or .docx files (such as Microsoft Word, StarOffice, or OpenOffice.org).

C#

```
bool bOpenInBrowser = false;
wt.Save(Page.Response, "StringVarOutput.doc", bOpenInBrowser);
```

VB.NET

```
Dim bOpenInBrowser As Boolean = False
wt.Save(Page.Response, "StringVarOutput.doc", bOpenInBrowser)
```

Alternatively, stream the file to the browser by passing Response.Stream to the Save method. In this case, to open the generated file in the default application for Word files:

1. Set the response's content-type header to "application/vnd.ms-word":

C#

```
Response.ContentType = "application/vnd.ms-word";
```

VB.NET

```
Response.ContentType = "application/vnd.ms-word"}}
```

2. Set the response's content-disposition to "attachment":

C#

```
Response.AddHeader("Content-Disposition",
     "attachment;filename=/"WordWriter.doc/"");
```

VB.NET

```
Response.AddHeader("Content-Disposition",\_
     "attachment;filename=""WordWriter.doc""")
```

3. Pass the Save method Response.Stream as a parameter:

C#

```
wt.Save(Response.OutputStream);
```

VB.NET

```
wt.Save(Response.OutputStream)
```

Save and Stream to Browser

You can call Save more than once for a single instance of WordTemplate. In the following examples Save is called twice:

1. `oWW.Save(Server.MapPath("./SaveMultipleOutput.doc")`

   Saves the generated Word file on the server.

2. `oWW.Save(Page.Response, "SaveMultipleOutput.doc", False)`

   Streams the generated Word file to the client. Since the parameter OpenInBrowser is False, the file will open in the default application for .doc files.

Pass the Template to WordApplication

You can use WordTemplate to open and populate a WordWriter template, then pass the populated workbook to WordApplication for additional processing. In this case, do not call Save. Instead, pass the WordTemplate object to WordApplication's Open method:

```
Document doc WordApplication.Open(WordTemplate template);
```

Example

C#

```
WordTemplate wTempl = new WordTemplate();
wTempl.Process();

WordApplication wApp = new WordApplication();
Document doc = wApp.Open(wTempl);
```

VB.NET

```
Dim wTempl As New WordTemplate
wTempl.Process()

Dim wApp As New WordApplication
Dim doc As Document = wApp.Open(wTempl)
```

Adding Formulas to a Document

Beginning in WordWriter version 4.1, WordTemplate now has the ability to evaluate a certain number of formulas on data fields that are brought into a template.

> ⚠ Word formulas are only supported with the Word 2007 OOXML file format (.docx/.docm).

Code Sample: WordTemplate Formula Demo

A WordTemplate formulas sample will be available shortly from the SoftArtisans OfficeWriter Demo Center. Screenshots from the sample are used below.

Placing Formulas in Merge Fields

Formulas in a Merge Field in your template can be used to calculate average, products, sums, and other formulas based on values in the data fields being bound to the template.

To place a formula into your template document, insert a merge field with a value as following:

```
<<=FORMULA(DataSource.Field)>>
```

The following formulas are valid with WordTemplate and will be evaluated when the data are bound to the document: AVERAGE, COUNT, COUNTA, MAX, MIN, PRODUCT, STDEV, STDEVP, SUM, VAR, and VARP. Additional information is available on the Creating Merge Fields page.

The WordTemplate Formula Demo uses formulas to display the average, standard deviation, maximum, and minimum college entrance exam scores of a group of fictional students as part of a table including their individual scores. A repeat block on the first row allows WordTemplate to bring in all of the values of the referenced fields.

Test date: «TestDate»
Test location: «Town»
Number of students: «=COUNTA(Items.StudentID)»

Student ID	Reading	Writing	Math	Total
«StudentID»	«ReadingScore»	«WritingScore»	«MathScore»	«TotalScore»
Avg	«=AVERAGE(Items.ReadingScore)»	«=AVERAGE(Items.WritingScore)»	«=AVERAGE(Items.MathScore)»	«=AVERAGE(Items.TotalScore)»
Stdev	«=STDEV(Items.ReadingScore)»	«=STDEV(Items.WritingScore)»	«=STDEV(Items.MathScore)»	«=STDEV(Items.TotalScore)»
High	«=MAX(Items.ReadingScore)»	«=MAX(Items.WritingScore)»	«=MAX(Items.MathScore)»	«=MAX(Items.TotalScore)»
Low	«=MIN(Items.ReadingScore)»	«=MIN(Items.WritingScore)»	«=MIN(Items.MathScore)»	«=MIN(Items.TotalScore)»

The following code opens the template and binds data to it using the SetDataSource and SetRepeatBlock methods, then streans the resulting document to the user:

```
protected void OnDownload(object sender, EventArgs e)
{
     WordTemplate wt = new WordTemplate();
     wt.Open(Page.MapPath("templates/TestScoreReport.docx"));

     string[] fields = new string[] { "TestDate", "School", "Town", "Version" };
     object[] values = new object[] {
     new DateTime(2009, 11, 16).ToShortDateString(),
          "Samuel Adams HS",
          "Milbury",
          wt.Version
          };
     wt.SetDataSource(values, fields, "");

     DataTable dt = GetScores();
     wt.SetRepeatBlock(dt, "Items");

     wt.Process();
     wt.Save(Response, "ScoreReport.docx", false);
}

private DataTable GetScores()
{
     DataTable dt = new DataTable();

     dt.Columns.Add("StudentID", typeof(string));
     dt.Columns.Add("ReadingScore", typeof(int));
     dt.Columns.Add("WritingScore", typeof(int));
     dt.Columns.Add("MathScore", typeof(int));
     dt.Columns.Add("TotalScore", typeof(int));

     Random rand = new Random();

     int rScore, wScore, mScore;
     for (int i = 0; i < 30; ++i)
     {
          rScore = rand.Next(400, 800);
          wScore = rand.Next(400, 800);
          mScore = rand.Next(400, 800);
          dt.Rows.Add(
               string.Format("S{0:000000}", rand.NextDouble() * 1000000),
               rScore, wScore, mScore, rScore + wScore + mScore
               );
     }

     DataTable dtSorted = dt.Clone();
     foreach (DataRow row in dt.Select("", "StudentID asc"))
          dtSorted.ImportRow(row);

     return dtSorted;
}
```

The formulas are evaluated by WordTemplate, as you can see in the output file:

Test date:	11/16/2009
Test location:	Milbury
Number of students:	30

Student ID	Reading	Writing	Math	Total
S000972	517	609	690	1816
S012286	690	687	561	1938
S019944	482	516	463	1461
S038044	400	677	725	1802
S144528	510	700	515	1725
S152202	776	749	621	2146
S268508	735	520	785	2040
S268629	454	631	499	1584
S268696	787	548	501	1836
S289371	428	762	417	1607
S291744	637	446	652	1735
S294071	520	421	599	1540
S322061	659	606	653	1918
S419318	561	533	564	1658
S469662	499	561	688	1748
S613649	793	447	697	1937
S619757	622	617	461	1700
S627363	595	770	452	1817
S658488	456	456	427	1339
S676564	488	712	457	1657
S694787	484	493	684	1661
S767279	788	781	431	2000
S786398	741	499	662	1902
S799806	757	578	694	2029
S812340	481	693	499	1673
S838011	416	756	517	1689
S879081	581	427	718	1726
S957026	644	776	730	2150
S975064	495	578	456	1529
S988330	620	636	479	1735
Avg	587.2	606.2	576.6	1769.9
Stdev	125.4	115.4	113.0	194.9
High	793	781	785	2150
Low	400	421	417	1339

Inserting an Image

WordWriter allows you to insert jpg, gif, bmp, and png images in merge fields. You can insert images repeat block and main document merge fields. There are two ways to insert an image into a merge field:

- Using a Placeholder
- Using the Image Modifier

Using a Placeholder

To insert an image in a merge field using a placeholder:

1. Create or open a WordWriter template (for instructions, see Creating a WordWriter Template).
2. Place the cursor within a merge field.
3. Open the **Insert** menu and select **Picture -> From File...**
4. Select a jpg, gif, bmp, or png image to insert in the merge field. (WordWriter does not support other image formats.)
5. Optional: Resize the image.
6. Save the template.

If you are using a database as a data source, the column that corresponds to the merge field containing the image must be of type BLOB ("Image" in SQL Server). If the data source is an array of objects, the value that corresponds to the merge field containing the image should be a simple byte array.

To specify how WordWriter should scale the image:

1. Right-click the image and select **Format Picture...**
2. Select the **Size** tab.
3. Check or uncheck **Lock aspect ratio**.
4. Check or uncheck **Relative to original picture size** .

Relative to original picture size	Lock aspect ratio	Meaning
false	false	The image will maintain its natural size and aspect ratio.
true	false	The image will be scaled to fit into the placeholder image but keep its natural aspect ratio.
false	true	The image will be scaled to fit into the placeholder image but keep its natural aspect ratio. The image will not be enlarged if it is smaller than the placeholder image.
true	true	The image will be scaled to fit into the placeholder image and its aspect ratio.

Using the Image Modifier

To insert an image in a merge field the "image" modifier:

1. Create or open a WordWriter template (for instructions, see Creating a WordWriter Template).
2. Add the "image" modifier in parentheses to the merge field in which you want to insert the image. For example, change <<Product.Image>> to <<Product.Image(image)>>
3. Optional: Set the "image" modifier's sizing arguments. For example, <<Product.Image(image(1,600,600))>> . The second and third parameters specify width and height, respectively, in twips. You can also use standard abbreviations to specify centimeters (cm), inches (in), or points (pt) when you use a template with the Office Open XML (.docx) format. The first argument tells WordWriter how to scale the image:

Value	Meaning
1	The image will always be the width and height specified regardless of its natural size.
2	The image will always be enlarged or shrunk to fit within the bounds specified by width and height but it will keep its natural aspect ratio.
3	The image will be scaled down to within the bounds specified by width and height but it will keep its natural aspect ratio.

If you are using a database as a data source, the column that corresponds to the merge field containing the image must be of type BLOB ("Image" in SQL Server). If the data source is an array of objects, the value that corresponds to the merge field containing the image should be a simple byte array.

Formatting Merge Fields

When you create a merge field in Microsoft Word (see Creating a WordWriter Template), you can apply formatting to the field. For example, you can set a merge field's format to "Uppercase" to display the field's value in uppercase letters. WordWriter supports most text, date, and time formats available in Microsoft Word.

- General Text Formatting Options
- Date and Time Formatting Options

> This section explains how to apply merge field formatting by adding format switches to the field codes. You can also apply all standard formatting (bold, italic, underline, etc.) to merge fields from Word's Format menu or toolbar.

General Text Formatting Options

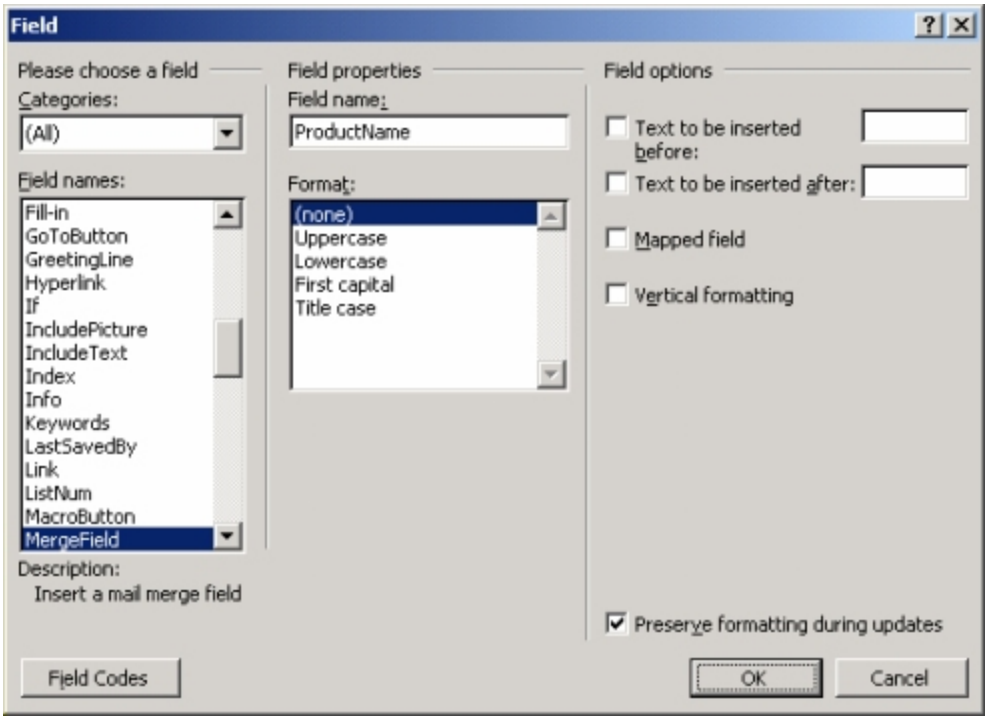
The Field dialog opens when you add a new merge field to a template. To open the dialog for an existing merge field, right-click the field and select **Edit Field...**

The following text formatting options can be set within Microsoft Word's Field dialog, or by entering formatting codes within the merge field:

- **Uppercase**

Displays the merge field's value in uppercase letters.

- **Lowercase**
  Displays the merge field's value in lowercase letters.
- **First capital**
  Capitalizes the first letter of the first word in the field.
- **Title case**
  Capitalizes the first letter of each word in the field.
- **Text Before**
  Inserts a specified phrase before the merge field value.
- **Text After**
  Inserts a specified phrase after the merge field value.



The Field Dialog in Microsoft Word 2002 (XP)

To set a merge field's formatting to Uppercase, Lowercase, Title case, or First capital using the Field dialog:

- The Field dialog opens automatically when you insert a new merge field. If you are formatting an existing merge field, right-click the merge field and select **Edit Field...**
- From the **Format** list select a format. For example, select **Uppercase**.

To display text before and/or after a merge field value:

- The Field dialog opens automatically when you insert a new merge field. If you are formatting an existing merge field, right-click the merge field and select **Edit Field...**
- Check **Text to be inserted before** and/or **Text to be inserted after**.
- In the Text Before text box, enter a phrase to display before the merge field value, and/or, in the Text After text box, enter a phrase to display after the merge field value.

You can also set these formatting options by entering special codes in the merge field. To view and edit a merge field's codes, right-click the field and select **Toggle Field Codes**. A field without special formatting looks like this:

```
{MERGEFIELD FieldName}
```

Format	Field Codes	Example

Uppercase	`{MERGEFIELD _FieldName_ \\* Upper}`	`{MERGEFIELD ProductName \\* Upper}`
Lowercase	`{MERGEFIELD _FieldName_ \\* Lower}`	`{MERGEFIELD ProductName \\* Lower}`
First capital	`{MERGEFIELD _FieldName_ \\* FirstCap}`	`{MERGEFIELD ProductName \\* FirstCap}`
Title case	`{MERGEFIELD _FieldName_ \\* Caps}`	`{MERGEFIELD ProductName \\* Caps}`
Text Before	`{MERGEFIELD _FieldName_ \b "_Text to insert before_"}`	`{MERGEFIELD OrderDate \b "Order Date: "}`
Text After	`{MERGEFIELD _FieldName_ \f "_Text to insert after_"}`	`{MERGEFIELD OrderDate \f " (Order Date)"}`

Date and Time Formatting Options

Using field codes, you can apply date/time formatting to merge fields that will bind to database fields of types Date, Time, and Timestamp, or to an object of type System.DateTime. The general format for applying date/time codes is:

```
{MERGEFIELD FieldName \@ "Formatting Pattern"}
```

Date/Time Format Code	Description	Field Code Example	Output Example
Day			
d	Displays the day of the month as a number. Single digit day numbers will not include a leading zero.	`{MERGEFIELD Date \@ "M/d/yyyy"}`	4/7/2003
dd	Displays the day of the month as a number. Single digit day numbers will include a leading zero.	`{MERGEFIELD Date \@ "dd MMMM yyyy"}`	07 April 2003

ddd	Displays a three-letter abbreviation for the day of the week.	`{MERGEFIELD Date \@ "ddd, d-MMM-yy"}`	Mon, 7-Apr-03
dddd	Displays the full name of the day of the week.	`{MERGEFIELD Date \@ "dddd, MMMM d, yyyy"}`	Monday, April 7, 2003
Month (Use uppercase M for months, lowercase m for minutes)			
M	Displays the month as a number. Single digit month numbers will not include a leading zero.	`{MERGEFIELD Date \@ "M/d/yy"}`	4/7/03
MM	Displays the month as a number. Single digit month numbers will include a leading zero.	`{MERGEFIELD Date \@ "yyyy-MM-dd"}`	2003-O4-07
MMM	Displays a three-letter month abbreviation.	`{MERGEFIELD Date \@ "MMM-yy"}`	Apr-03
MMMM	Displays the full month name	`{MERGEFIELD Date \@ "MMMM d, yyyy"}`	April 7, 2003
Year			
yy	Displays a two-digit year number. For years 1-9, the year number will include a leading zero.	`{MERGEFIELD Date \@ "MMM-yy"}`	Apr-03
yyyy	Displays a four digit year number	`{MERGEFIELD Date \@ "MMMM d, yyyy"}`	April 7, 2003
Hours			

h	Use lowercase h to display times based on the 12-hour clock. Lowercase h displays single-digit hours as single-digit numbers, without a leading zero.	`{MERGEFIELD Time \@ "h:mm"}`	6:05
hh	Use lowercase hh to display times based on the 12-hour clock. Lowercase hh displays single-digit hours with a leading zero.	`{MERGEFIELD Time \@ "hh:mm am/pm"}`	06:05 am
H	Use uppercase H to display times based on the 24-hour clock. Uppercase H displays single-digit hours as single-digit numbers, without a leading zero.	`{MERGEFIELD Time \@ "H:mm"}`	7:05
HH	Use uppercase HH to display times based on the 24-hour clock. Uppercase HH displays single-digit hours with a leading zero.	`{MERGEFIELD Time \@ "HH:mm"}`	18:30
Minutes (Use lowercase m for minutes, uppercase M for months)			
m	Displays single-digit minutes without a leading zero.	`{MERGEFIELD Time \@ "m 'minutes'"}`	2 minutes
mm	Displays single-digit minutes with a leading zero.	`{MERGEFIELD Time \@ "hh:mm am/pm"}`	06:05 am
AM & PM			
AM/PM	Displays uppercase AM or PM.	`{MERGEFIELD Time \@ "hh:mm AM/PM"}`	08:30 PM
am/pm	Displays lowercase am or pm.	`{MERGEFIELD Time \@ "hh:mm am/pm"}`	07:55 am
Other Text			

'text'	To display text within a date or time, enclose the text in single quotation marks.	`{MERGEFIELD Date \@ "hh:mm 'EST'"}`	06:15 EST
character	To include a character, such as - (hyphen), in a date or time, do not enclose in quotation marks.	`{MERGEFIELD Date \@ "MMM-yy"}`	Apr-03

WordApplication in Depth

- Creating a Paragraph
- Formatting Text
- Creating Lists
- Creating Tables
- Creating Sections
- Adding Images
- Inserting Watermarks
- Importing Data
- WordApplication Output Options

Creating a Paragraph

To insert a paragraph in a section of your document, use one of the following methods. All of the methods insert a parapraph at a specified point in an Element. An Element object represents an editable region in a document, such as a section or a table cell. The document itself is an Element (the Document class extends Element). Later we will add a paragraph to a Document object.

- `Paragraph Element.InsertParagraphAfter(NamedStyle style)`

  Inserts an empty paragraph at the end of the specified Element. This method applies the specified named style to the paragraph.

- `Paragraph Element.InsertParagraphAfter(NamedStyle style, ParagraphFormatting props)`

  Inserts an empty paragraph at the end of the specified Element. This method applies a specified named style and a set of formatting properties to the paragraph. The formatting properties will override the style's properties

- `Paragraph Element.InsertParagraphBefore(NamedStyle style)`

  Inserts an empty paragraph at the beginning of the specified Element. This method applies the specified named style to the paragraph.

- `Paragraph Element.InsertParagraphBefore(NamedStyle style, ParagraphFormatting props)`

  Inserts an empty paragraph at the beginning of the specified Element. This method applies a specified named style and a set of formatting properties to the paragraph. The formatting properties will override the style's properties

- `Paragraph Position.InsertParagraph(NamedStyle style)`

  Inserts an empty paragraph at a specified position within an Element. A Position object represents a cursor. This method applies the specified named style to the paragraph.

Before adding a paragraph to your file, you must create a content region in which to insert the paragraph. Each editable region in a Word file is represented by an Element object, or an object that extends Element, such as a Document object.

```
WordApplication wordApp = new WordApplication();
Document doc = wordApp.Create();
```

To insert an empty paragraph, use one of the methods listed at the beginning of this section. All of the methods take a style argument, which specifies a named style to apply to the inserted paragraph. The style argument may be null. If it is null, the Normal style will be applied to the paragraph. The Normal style is the default style used by Microsoft Word. The following inserts an empty paragraph at the end of the document and applies the BodyText style to the paragraph:

```
Paragraph para = doc.InsertParagraphAfter(doc.Styles[NamedStyle.BuiltIn.BodyText]);
```

To add text to the paragraph, call one of the Element class's InsertTextAfter or InsertTextBefore methods. You can apply a font to the text by passing a Font object to InsertTextAfter or InsertTextBefore. The following lines create a Font object that we will use later when adding text to the paragraph:

```
Font paragraphFont = doc.CreateFont();
paragraphFont.FontName = "Times New Roman";
paragraphFont.FontSize = 10;
```

The method InsertTextAfter takes a string and inserts it at the end of the paragraph. The method's second parameter specifies a Font object (created above) to apply to the text.

```
para.InsertTextAfter("This document was created with OfficeWriter for Word. It
     contains a single, left-justified paragraph.",
     paragraphFont);
```

The Paragraph class provides numerous formatting properties. The following left-justifies the paragraph:

```
//--- If para is a Paragraph object
para.Formatting.TextJustification = ParagraphFormatting.Justification.Left;

//--- If myFormat is a ParagraphFormatting object
myFormat.TextJustification = ParagraphFormatting.Justification.Left;
```

You can access existing paragraphs through the Element.Elements property, for example:

```
//--- Open an existing Word file and get the first
//--- paragraph.
WordApplication wordApp = new WordApplication();
Document doc = wordApp.Open(@"C:\sample.doc");
Paragraph firstParagraph = (Paragraph)doc.GetElements(Element.Type.Paragraph)[0];
```

Paragraph Formatting

To apply a font to text in a paragraph, create a Font object and pass it to Paragraph.InsertTextAfter or Paragraph.InsertTextBefore, as demonstrated above.

Other paragraph formatting can be applied through the Paragraph object or the ParagraphFormatting object. To create a ParagraphFormatting object, call Document.CreateParagraphFormatting:

```
WordApplication wordApp = new WordApplication();
Document doc = wordApp.Create();
ParagraphFormatting bodyTextFormatting = doc.CreateParagraphFormatting();
```

Next, set ParagraphFormatting properties.

```
//--- Set the amount of whitespace that should be
//--- placed above a paragraph in twips (1/1440 in).
bodyTextFormatting.SpaceBefore = 240;

//--- Display line numbers.
bodyTextFormatting.LineNumberingAllowed = true;
```

To assign the ParagraphFormatting object that you created to a paragraph, pass it to Element.InsertParagraphAfter or Element.InsertParagraphBefore when you create a new paragraph.

```
Paragraph para = doc.InsertParagraphAfter(doc.Styles[NamedStyle.BuiltIn.BodyText],
     bodyTextFormatting);
```

Formatting Text

- Applying Fonts
- Named Styles
- Paragraph Formatting
- Table Formatting

Applying Fonts

Using the Font class, you can apply text formatting to a CharacterRun or HTMLToWord object. To return a Font object use:

- `Document.CreateFont`

Returns a Font object that represents the font used by the document's Normal style.
Or

- `NamedStyle.Font`

Returns a copy of the specified named style's font.

To define a font and apply it to a paragraph, first create a Font object:

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
Font paragraphFont = doc.CreateFont();
paragraphFont.FontName = "Times New Roman";
paragraphFont.FontSize = 10;
```

Add an empty paragraph to your document:

```
//--- InsertParagraphAfter's style argument may be null.  If it
//--- is null, the Normal style will be applied to the paragraph. The
//--- Normal style is the default style used by Microsoft Word.
Paragraph para = doc.InsertParagraphAfter(null);
```

To add text to the paragraph call one of the Element class's InsertTextAfter or InsertTextBefore methods. Pass the the text to add and your Font object to the method:

```
//--- The following method takes a string and inserts it
//--- at the end of the paragraph.  The method's second
//--- parameter specifies a Font object to apply to the
//--- text.
para.InsertTextAfter("This document was created with
     OfficeWriter for Word.",
     paragraphFont);
```

To get a copy of a built-in style's font, use the NamedStyle.Font property.

```
NamedStyle heading1Style = doc.Styles[NamedStyle.BuiltIn.Heading1];
Font heading1Font = heading1Style.Font;
```

Named Styles

The Styles collection contains all the named styles in the Word document. All Word documents contain a set of built-in styles. These are included in the Styles collection. If you use WordWriter to open an existing Word file, it may also contain custom named styles which will be added to the Styles collection.

You cannot create or modify a named style with WordWriter.

To return a document's Styles collection, use the property Document.Styles:

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
NamedStyle heading1Style = doc.Styles[NamedStyle.BuiltIn.Heading1];
```

To apply a style to a paragraph, pass it to InsertParagraphAfter or InsertParagraphBefore when you add the paragraph to your document:

```
Paragraph para =
     doc.InsertParagraphAfter(doc.Styles[NamedStyle.BuiltIn.BodyText]);
```

Paragraph Formatting

When you add a paragraph to a document using Element.InsertParagraphAfter or Element.InsertParagraphBefore, you can pass a ParagraphFormatting object to the method. A ParagraphFormatting object contains a set of formatting properties that can be applied to a paragraph.

To return a ParagraphFormatting object, use:

- `Document.CreateParagraphFormatting`

Returns a ParagraphFormatting object that represents the font used by the document's BodyText style.
Or

- `NamedStyle.ParagraphFormatting`

Returns a copy of the specified named style's paragraph formatting properties.//--- Return the paragraph formatting of the BodyText style.

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
ParagraphFormatting bodyTextFormatting = doc.CreateParagraphFormatting();

//--- Return a copy of the paragraph formatting of the BodyText2 Style.
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
ParagraphFormatting bodyText2Format =
     doc.Styles[NamedStyle.BuiltIn.BodyText2].ParagraphFormatting;
```

To assign the ParagraphFormatting object that you created to a paragraph, pass it to Element.InsertParagraphAfter or Element.InsertParagraphBefore when you create a new paragraph.

```
Paragraph para = doc.InsertParagraphAfter(doc.Styles[NamedStyle.BuiltIn.BodyText],
     bodyTextFormatting);
```

Table Formatting

To apply a font to text in a table cell, create a Font object and pass it to TableCell.InsertTextAfter or TableCell.InsertTextBefore, as demonstrated above.

Other paragraph formatting can be applied through the Table object or the TableFormatting object. To create a TableFormatting object, call Document.CreateTableFormatting:

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
TableFormatting tableFormat = doc.CreateTableFormatting();
```

Next, set TableFormatting properties.

```
tableFormat.DefaultShading.BackgroundColor = Color.Gray;
tableFormat.SetDefaultPadding(TableCell.Location.Top, 720);
```

To assign the TableFormatting object that you created to a table, pass it to Element.InsertTableAfter or Element.InsertTableBefore when you create a new table.

```
Table tbl = doc.InsertTableAfter(3, 5, tableFormat);
```

Creating Lists

Lists

To add a list to a section of your document, use one of the following methods. All of the methods insert a list at a specified point in an Element. An Element object represents an editable region in a document, such as a section or a table cell. The document itself is an Element (the Document class extends Element). Later, we will add a list to a Document object.

- `List Element.InsertListAfter(boolean numbered)`

Inserts an empty list at the end of the specified Element. Pass true to the method to create a numbered list, and false to create a bulleted list.

- `List Element.InsertListBefore(boolean numbered)`

Inserts an empty list at the beginning of the specified Element. Pass true to the method to create a numbered list, and false to create a bulleted list.

- `List Position.InsertList(boolean numbered)`

  Inserts an empty list at a specified position within an Element. A Position object represents a cursor. Pass true to the method to create a numbered list, and false to create a bulleted list.

Before adding a list to your file, you must create a content region in which to insert the list. Each editable region in a Word file is represented by an Element object, or an object that extends Element, such as a Document object.

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
```

To insert an empty list, use one of the methods listed at the beginning of this section, for example:

```
//--- The boolean parameter specifies whether the
//--- list is numbered (true) or bulleted (false).
List numberedList = doc.InsertListAfter(true);
```

Next, create list entries. To add a new entry, call either List.AddEntry or List.InsertEntry.

```
//--- AddEntry's parameter specifies the 0-based
//--- indent level at which to insert the new entry. A list
//--- may contain up to 9 levels, so the deepest level
//--- is 8.  The following inserts the first list entry, at
//--- the top list level.
ListEntry firstListEntry = numberedList.AddEntry(0);

//--- InsertEntry's first parameter specifies the 0-based
//--- position in the list at which to insert the new entry
//--- (for the first entry this parameter should be O,
//--- for the second 1, and so on).  The second parameter
//--- specifies the 0-based level of the new entry.  A list
//--- may contain up to 9 levels, so the deepest level
//--- is 8.  The following inserts the second list entry, at
//--- the top list level.
ListEntry secondListEntry = numberedList.InsertEntry(1, 0);
```

To add text to a list entry call one of the Element class's InsertTextAfter or InsertTextBefore methods. You can apply a font to the text by passing a Font object to InsertTextAfter or InsertTextBefore. The following lines create a Font object that we will use later when adding text to the paragraph:

```
Font listFont = doc.CreateFont();
listFont.FontName = "Times New Roman";
listFont.FontSize = 10;
```

The method InsertTextAfter takes a string and inserts it at the end of the list entry. The method's second parameter specifies a Font object (created above) to apply to the text.

```
firstListEntry.InsertTextAfter("OfficeWriter for Word",
     listFont);
secondListEntry.InsertTextAfter("OfficeWriter for Excel",
     listFont);
```

You can access existing lists through the Element.Elements property.

```
//--- Open an existing Word file and get the first
//--- table.
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Open(@"C:\sample.doc");
List firstList = doc.GetElements(Element.Type.List)[0];
```

See Also

List

ListEntry

ListLevel

Creating Tables

To add a table to a section of your document, use one of the following methods. All of the methods insert a table at a specified point in an Element. An Element object represents an editable region in a document, such as a section or paragraph. The document itself is an Element.

- `Table Element.InsertTableAfter(int rows, int columns)`

  Inserts an empty table, with a specified number of rows and columns, at the end of the element.

- `Table Element.InsertTableAfter(int rows, int columns, TableFormatting props)`

  Inserts an empty table, with a specified number of rows and columns, at the end of the element. This method applies a set of table formatting properties (contained in a TableFormatting object) to the new table.

- `Table Element.InsertTableBefore(int rows, int columns)`

  Inserts an empty table, with a specified number of rows and columns, at the beginning of the element.

- `Table Element.InsertTableBefore(int rows, int columns, TableFormatting props)`

  Inserts an empty table, with a specified number of rows and columns, at the beginning of the element. This method applies a set of table formatting properties (contained in a TableFormatting object) to the new table.

- `Table Position.InsertTable(int rows, int columns)`

  Inserts an empty table at a specified position within an Element. A Position object represents a cursor.

Before adding a table to your file, you must create a content region in which to insert the table. Each editable region in a Word file is represented by an Element object, or an object that extends Element, such as a Document object.

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
```

To insert an empty table, use one of the methods listed at the beginning of this section, for example:

```
//--- Create a table with 5 rows and 4 columns
//--- and insert it at the end of the document.
Table tbl = doc.InsertTableAfter(5, 4);
```

To enter text in the table, first return one or more TableCellobjects:

```
TableCell firstCell = tbl[1, 1];
TableCell secondCell = tbl[1, 2];
```

To add text to a table cell, call one of the Element class's InsertTextAfter or InsertTextBefore methods.

```
//--- When adding text to a table, you can
//--- specify a font for the text.  First, create a Font
//--- object:
Font cellFont = doc.CreateFont();
cellFont.FontName = "Times New Roman";
cellFont.FontSize = 10;

//--- The following method takes a string and inserts it
//--- at the end of the table cell.  The method's second
//--- parameter specifies a Font object to apply to the
//--- text.
firstCell.InsertTextAfter("OfficeWriter for Word",
   cellFont);
secondCell.InsertTextAfter("OfficeWriter for Excel",
   cellFont);
```

You can access existing tables through the Element.Elements property.

```
//--- Open an existing Word file and get the first
//--- table.
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Open(@"C:\sample.doc");
Table firstTable = (Table)doc.GetElements(Element.Type.Table)[0];
```

Table Formatting

To apply a font to text in a table cell, create a Font object and pass it to TableCell.InsertTextAfter or TableCell.InsertTextBefore, as demonstrated above.

Other paragraph formatting can be applied through the Tableobject or the TableFormatting object. To create a TableFormatting object, call Document.CreateTableFormatting:

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
TableFormatting tblFormatting = doc.CreateTableFormatting();
```

Next, set TableFormatting properties.

```
tblFormatting.DefaultShading.BackgroundColor = Color.Gray;
tblFormatting.set_DefaultPadding(TableCell.Location.Top, 720);
```

To assign the TableFormatting object that you created to a table, pass it to Element.InsertTableAfter orElement.InsertTableBefore when you create a new table.

```
Table tbl = doc.InsertTableAfter(3, 5, tblFormatting);
```

Creating Sections

A Section object represents a major section in a document, like a chapter in a book. Many documents will only contain one section. In more complex documents, content can be divided into multiple sections.

You can assign a different set of formatting settings to each section, including:

- Margins
- Paper size or orientation
- Paper source for a printer
- Page borders
- Vertical alignment
- Headers and footers
- Columns
- Page numbering

- Line numbering
- Footnotes and endnotes

To add a section to your document, use one of the following methods. The methods insert a section at the beginning or end an Element. An Element object represents an editable region in a document, such as a section or a paragraph. The document itself is an Element.

- `Section Element.CreateSectionAfter()`

  Inserts an empty section at the end of the specified Element.

- `Section Element.CreateSectionBefore()`

  Inserts an empty section at the beginning of the specified Element.

Before adding a section to your file, create a Document object.

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
```

Call Document.CreateSectionAfter or Document.CreateSectionBefore to add a section to the document

```
//--- Create a section at the end of the document.
Section firstSection = doc.CreateSectionAfter();
```

Next, set Section properties, for example:

```
//--- Add a footer to the section.  A header or
//--- footer can be applied to all pages in a section,
//--- the first page in a section, odd pages only, or
//--- even pages only.  The following returns a footer
//--- for all pages in the section.
Element footer = firstSection.GetFooter(Section.HeaderFooterType.All);
footer.InsertTextAfter("Copyright 2007 SoftArtisans Inc.", true);

//--- Set the width of a margin in twips (1/1440 in).
firstSection.LeftMargin = 1440;
firstSection.RightMargin = 1440;
```

After creating a Section and setting Section properties, add content. For example, add a paragraph:

```
Paragraph para = firstSection.InsertParagraphAfter
     (doc.Styles[NamedStyle.BuiltIn.Normal]);
para.InsertTextAfter("OfficeWriter is now fully integrated and compatible
     with SQL Server Reporting Services. Business users
     can now create and publish their Reporting Services
     reports without ever leaving Excel and Word. With
     OfficeWriter, every feature of Excel and Word is
     preserved in Reporting Services.",
     true);
```

You can access existing sections through the Document.Sections property.

```
//--- Open an existing Word file and get the first
//--- section.
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Open(@"C:\sample.doc");
Section firstSection = doc.Sections[0];
```

Adding Images

To insert an image into a section of your document, use one of the following methods. Each method inserts an image at the beginning or end of an Element. An Element object represents an editable region in a document, such as a section or paragraph. The document itself is an Element.

- `InlineImage Element.InsertImageBefore(String fileName)`

  Inserts the image specified by file name at the beginning of the element.

- `InlineImage Element.InsertImageAfter(String fileName)`

  Inserts the image specified by file name at the end of the element.

- `InlineImage Element.InsertImageBefore(System.IO.Stream stream)`

  Inserts the image contained in the specified stream at the beginning of the element.

- `InlineImage Element.InsertImageAfter(System.IO.Stream stream)`

  Inserts the image contained in the specified stream at the end of the element.

Before adding an image to your file, you must create a content region in which to insert the image. Each editable region in a Word file is represented by an Element object, or an object that extends Element, such as a Document

```
object.WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
```

To insert an image, use one of the methods listed at the beginning of this section, for example:

```
InlineImage img = doc.InsertImageAfter("C:\\sample.jpg");
```

An inserted image is returned as an InlineImage object. The InlineImage class extends Element.

Inserting Watermarks

A watermark is an image or text that when printed appears in the background of a document. The watermark text or image is faded so that it does not distract from the content of the document but is legible.

The property Document.Watermark returns a Watermark object.

```
WordApplication app = new WordApplication();
Document doc = app.Create();
Watermark mark = doc.Watermark;
```

To create a text watermark, assign the string that you want to display in the background of the document to the Watermark object's Text property.

```
Watermark mark = doc.Watermark;
mark.Text = "CONFIDENTIAL";
mark.WatermarkLayout = Watermark.Layout.Diagonal;
```

To use an image as a watermark, assign an image file to the Watermark object's Image property.

```
Watermark mark = doc.Watermark;
mark.Image = File.Open("somepic.gif", FileMode.Open);
```

To remove a watermark from a document, call the Watermark object's ClearWatermark method.

```
Document doc = app.Create();
Watermark mark = doc.Watermark;
mark.ClearWatermark();
```

Importing Data

WordApplication's ImportData method allows you to import blocks of data to a worksheet from a database or a rectangular array. ImportData is a method of both Element and Table. Element.importData returns a Table object representing a table in the document that contains the imported data.Table.importData adds an new row to an existing table.

- Importing from a Database
- Importing from a Rectangular Array
- Customizing Your Data Import

Importing from a Database

You can import values from a database to a table in your document by passing the ImportData method a DataTable, DataView, SqlDataReader, OleDbDataReader, or AdomdDataReader.

The DataTable and DataView classes are in the System.Data namespace. Use an Import directive to import the namespace to the aspx page:

```
<%@ Import namespace="System.Data" %>
```

To import System.Data to a C# code-behind page (.aspx.cs), use:

```
using System.Data;
```

To import database values using OleDb, import the System.Data.OleDb namespace to your page. To import database values using SqlClient, import the System.Data.SqlClient namespace to your page.

To import values from a database to your document:

1. Connect to the database and execute a query to return a DataTable, DataView, SqlDataReader, OleDbDataReader, or AdomdDataReader, for example:

```
private DataTable GetEmployeeDataTable()
{
     string employeeSQL = "SELECT TOP 10 FirstName + ' ' + LastName As Name, " +
          "Title FROM Employees";

     DataTable dt = new DataTable();
     using(SqlConnection conn = new SqlConnection(connString))
          new SqlDataAdapter(employeeSQL, conn).Fill(dt);

     return dt;
}
```

2. Create a document:

```
WordApplication wwApp = new WordApplication();
Document doc = wwApp.Create();
```

3. Pass the DataTable returned from the database to ImportData:

```
Table employeesTable = doc.ImportData(dt);
```

Importing from an Array

1. Create a rectangular array, for example:

```
string[,] arrayData = {{"Nancy", "Davolio", "Sales Manager"},
     {"Michael", "Suyama", "HR Representative"},
     {"Adrian", "King", "IS Support"}};
```

2. Create a document:

```
WordApplication wwApp = new WordApplication();
Document doc = wwApp.Create();
```

3. Pass the array to ImportData:

```
Table employeesTable = doc.ImportData(arrayData);
```

Customizing Your Data Import

The DataImportProperties class contains a set of properties that are used when importing data to a table in a document. The settings of a DataImportProperties object will be applied to a data import if the object is passed to ImportData (with the set of values to import). You can create several DataImportProperties objects and assign a different one to each data import, or re-use one object in multiple ImportData calls.

To customize a data import using a DataImportProperties object:

1. Create a DataImportProperties object:

```
WordApplication wwApp = new WordApplication();
Document doc = wwApp.Create();
DataImportProperties importProps = doc.CreateDataImportProperties();
```

2. Set one or more data import properties:

```
//--- Import column names to the first row of the table.
importProps.UseColumnNames = true;

//--- Automatically resize the table's columns and rows to fit
//--- the imported data.
importProps.AutoFit = true;
```

3. Define a DataTable, DataView, SqlDataReader, OleDbDataReader, AdomdDataReader or rectangular array, for example:

```
OleDbConnection Conn = new OleDbConnection();
DataTable employeeDt = null;
int employeeId;

try
{
     Conn.ConnectionString = Application["connstring"].ToString();
     //--- SQL Query for employee information
     string employeeSQL = "SELECT FirstName + ' ' + LastName As Name, Title " +
         "FROM Employees WHERE employeeID=?";
     OleDbCommand cmdEmployee = new OleDbCommand(employeeSQL, Conn);
     cmdEmployee.Parameters.Add("@employeeID", employeeID);
     OleDbDataAdapter adptEmployee = new OleDbDataAdapter(cmdEmployee);
     employeeDt = new DataTable();
     adptEmployee.Fill(employeeDt);
}
```

4. Pass the data and the DataImportProperties object to ImportData:

```
Table importTable = doc.ImportData(employeeDt , importProps);
```

WordApplication Output Options

After creating a document with WordApplication, you can:

- Save the generated Word file to disk,
- Stream the Word file to the client,
- Write the Word file to a specified output stream, or,
- Pass the document to a WordTemplate object.

When WordWriter streams a file to the client, the output result will depend on the browser as well as the server-side script.

Save to Disk

To save the generated file on the server, call:

```
void Save(Document doc,
 bool preserve,
 String fileName)
```

If preserve is true, WordWriter will try to preserve all formatting and features in the file, including those that it does not directly support. These includes fields, footnotes, hyperlinks, comments, and anchored images. If preserve is false, only features directly supported by WordWriter will be preserved. These includes all tables, lists, headers, and footers. Setting preserve to false will produce a much smaller file.

fileName specifies a complete physical path and file name for the generated file. WordWriter will save the file to this location. If a file with the same name exists, it will be overwritten by the new Word file.

Example

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
wwapp.Save(doc, false, @"C:\Sales2003\June.doc");
```

Stream to Client

To stream the generated file to the client, call one of the following methods:

```
void Save(Document doc,
     bool preserve,
     System.Web.HttpResponse response,
     String fileName,
     bool openInBrowser)

void Save(Document doc,
     bool preserve,
     System.Web.HttpResponse response,
     String fileName,
     bool openInBrowser,
     String contentType)
```

If preserve is true, WordWriter will try to preserve all formatting and features in the file, including those that it does not directly support. These includes fields, footnotes, hyperlinks, comments, and anchored images. If preserve is false, only features directly supported by WordWriter will be preserved. These includes all tables, lists, headers, and footers. Setting preserve to false will produce a much smaller file.

fileName specifies a complete physical path and file name for the generated file. WordWriter will save the file to this location. If a file with the same name exists, it will be overwritten by the new Word file.

Browsers other than Internet Explorer cannot embed a Word file in the browser window. When the generated document is streamed to a browser other than IE, the user will always be prompted to open or save the file. If the user chooses to open the file, it will open in Microsoft Word or another document application.

Internet Explorer can display a Word file in the browser window. When the generated document is streamed to Internet Explorer, the browser's settings will determine whether the file opens automatically or the user is asked to open or save the file. The parameter openInBrowserdetermines whether the file will open in IE or in a document application. If openInBrowser is set to true, the response content-disposition header is set to open the file in the browser window.

Example

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
wwapp.Save(doc,
     false,
     Page.Response,
     "file.doc",
     true,
     "application/vnd.ms-word");
```

Write to Stream

To write the generated document to a System.IO.Stream, call:

```
void Save(Document doc,
     bool preserve,
     OutputStream out)
```

If preserve is true, WordWriter will try to preserve all formatting and features in the file, including those that it does not directly support. These includes fields, footnotes, hyperlinks, comments, and anchored images. If preserve is false, only features directly supported by WordWriter will be preserved. These includes all tables, lists, headers, and footers. Setting preserve to false will produce a much smaller file.

Example

```
WordApplication wwapp = new WordApplication();
Document doc = wwapp.Create();
System.IO.Stream fstream = new System.IO.FileStream (@"C:\file.doc", FileMode.Create);
wwapp.Save(doc, false, fstream);
```

Return a Template

You can use WordApplication to create a document with merge fields (i.e., a template) and pass the file to WordTemplate to populate the merge fields. To pass a document to WordTemplate, use the following method:

```
WordTemplate.Open(WordApplication aApp, Document aDoc)
```

Example

```
WordApplication wwApp = new WordApplication();
Document doc = wwApp.Create();
WordTemplate wwTmpl = New WordTemplate();
wwTmpl.Open(wwApp, doc);
```

SharePoint Integration in Depth

Introduced in build 4.1.0.1379

- Exporting list as Word document
- SharePoint Solution - Word Export Plus

Exporting list as Word document

Introduced in build 4.1.0.1379

- Quick start
- Step 1: Set up the SharePoint project
- Step 2: Provision custom list definition
- Step 3: Provision custom list instance
- Step 4: Provision template document library
- Step 5: Provision output document library
- Step 6: Create custom menu action
- Step 7: Create application page
- Step 8: Create the template
- Step 9: Managing the SharePoint solution package
- Step 10: Using the tutorial

Beginning with OfficeWriter 4.5, WordWriter offers integration with SharePoint. This tutorial shows you how to provision a SharePoint custom list, then export the list items as a Word document. To a lesser extent, this tutorial also illustrates how to create a grouped document using the WordTemplate object.

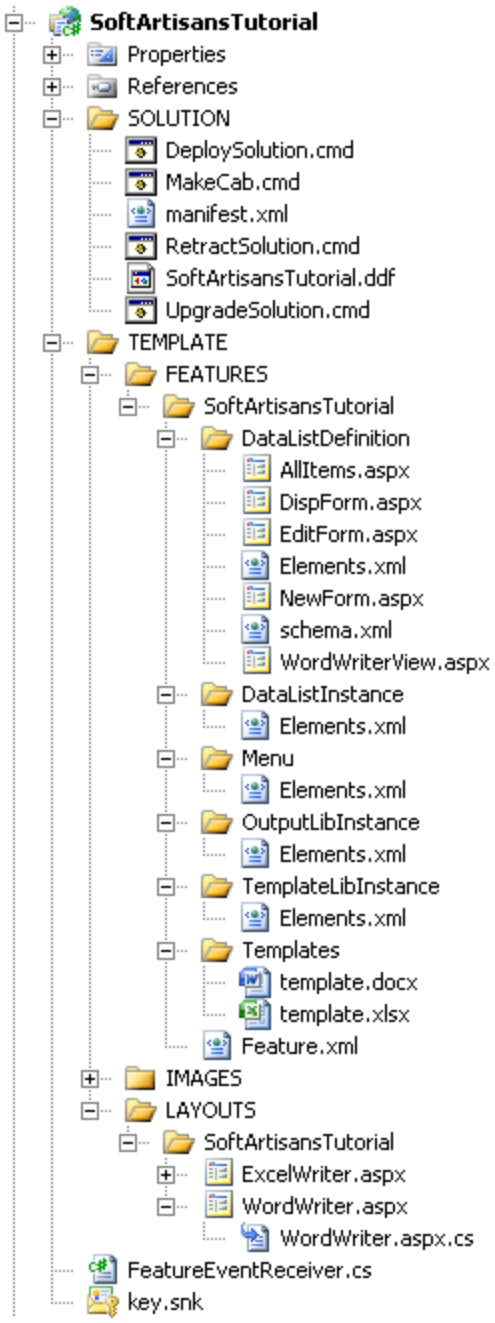
> The SharePoint integration and grouping features are only available in WordWriterEE. You must install WordWriter EE before running the tutorial.
>
> SharePoint integration makes use of extension methods, which require .NET 3.5. Additionally, the grouping and formula evaluation features used in this tutorial require the use of a Word 2007 OOXML file format (.docx/.docm) file.

Quick start

- Download and unpack the tutorial archive, which contains a complete Visual Studio 2008 project
- Follow the instructions in Step 9: Managing the SharePoint solution package
- Follow the instructions in Step 10: Using the tutorial

Step 1: Set up the SharePoint project

Download and unpack the tutorial archive, which contains a complete Visual Studio 2008 project. We set up the project directories to mirror those in SharePoint's 12 Hive. The same project is used for both ExcelWriter and WordWriter tutorials.



The project provisions the following:

- a custom list definition,
- an instance of the custom list containing sample data,
- a standard document library containing the WordWriter template,
- an empty document library to store WordWriter-generated document
- custom entries on the Action drop-down menu

While you can provision each of the above using a separate feature, in this tutorial we do so with a single feature in order to simplify deployment.

When deployed, the project installs the OfficeWriter DLLs into the Global Assembly Cache. Locate the OfficeWriter DLLs, which are installed by default in the *Program Files\SoftArtisans\OfficeWriter\bin\dotnet* folder. Copy the following OfficeWriter DLLs into the project's *bin* folder:

- SoftArtisans.OfficeWriter.ExcelWriter.dll
- SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll
- SoftArtisans.OfficeWriter.WordWriter.dll
- SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Furthermore, you must ensure that the versions referenced in the project match the versions that you have. Open the *manifest.xml* file in the *SOLUTION* folder of the project. If necessary, modify the version attribute of each OfficeWriter assembly to match the version of the DLL in the

project's *bin* folder. You can view the assembly version in the file properties of the DLL, under the Version tab.

```
<Assembly DeploymentTarget="GlobalAssemblyCache" Location="SoftArtisans.OfficeWriter.ExcelWriter.dll">
  <SafeControls>
    <SafeControl Assembly="SoftArtisans.OfficeWriter.ExcelWriter, Version=7.1.0.2018, Culture=neutral, PublicKe
  </SafeControls>
</Assembly>
<Assembly DeploymentTarget="GlobalAssemblyCache" Location="SoftArtisans.OfficeWriter.WordWriter.dll">
  <SafeControls>
    <SafeControl Assembly="SoftArtisans.OfficeWriter.WordWriter, Version=4.1.0.1490, Culture=neutral, PublicKey
  </SafeControls>
</Assembly>
```

Step 2: Provision custom list definition

The *DataListDefinition* folder contains the files needed to provision a custom list. The custom list definition is based on the standard SharePoint custom list type.

```
<ListTemplate
  Name="DataListDefinition"
  DisplayName="SoftArtisans Data List"
  Description="Custom list for sample data"
  BaseType="0"
  Type="999"  ——— must be unique
  OnQuickLaunch="true"
  SecurityBits="11"
  Sequence="410"
  Image="/_layouts/images/itgen.gif" />
```

Each list definition must provide a schema.xml file. Again for simplicity, we copy and modify the schema.xml file of the SharePoint custom list, which can be found in the *FEATURES\CustomList* folder under the 12 Hive.

The custom list contains orders placed with the fictitious PostOne store. Each item should have the following fields: order ID, order date, amount, customer name, address, city, U.S. state and U.S. zip code. Additionally, we add a calculated field, order year, which extracts the year from the order date; the order year is used for grouping. In the schema.xml file, the fields are specified under the <*Fields*> node.

```
<Fields>
  <Field ID="{2A577663-5936-4d77-AED7-DD74EBA09353}" Name="orderid" DisplayName="Order ID" Type="Text" Group="Soi
  <Field ID="{DB3D5156-3126-4d0b-9D89-1EDD3E3ECC66}" Name="orderdate" DisplayName="Order Date" Type="DateTime" Gr
  <Field ID="{BD1AABE6-0767-4722-8B57-7F8F499C4F5A}" Name="orderamount" DisplayName="Order Amount" Type="Currency
  <Field ID="{06887C4E-8344-49b4-8947-A4F0598A8B4C}" Name="customer" DisplayName="Customer" Type="Text" Group="Sc
  <Field ID="{01D0CF77-EACA-4f78-AE89-323E087D16FD}" Name="address" DisplayName="Address" Type="Text" Group="Soft
  <Field ID="{F4B94673-8B04-4bdf-A734-DADB2E65B4F6}" Name="city" DisplayName="City" Type="Text" Group="SoftArtise
  <Field ID="{CBA5D64D-79B9-4253-BE26-B8599E938C2B}" Name="state" DisplayName="State" Type="Text" Group="SoftArt:
  <Field ID="{5BD26A89-76EF-4cec-8F98-834510930AC5}" Name="zipcode" DisplayName="Zip Code" Type="Text" Group="Soi
  <Field ID="{AC0F3D1D-B0B4-44ca-9151-61E149A5FDB2}" Name="orderyear" DisplayName="Order Year" Type="Calculated"
    <Formula>=TEXT(YEAR([orderdate]),"000")</Formula>
    <Default />
    <FieldRefs>
      <FieldRef Name="orderdate" />
    </FieldRefs>
  </Field>
```

calculated field

To illustrate grouping, we add a custom view named *WordWriterView*. While we can copy an existing view definition and make edits manually, a much quicker method is to create the view using the SharePoint UI then copy its definition using the SharePoint Manager. The custom view specifies the fields to be shown under the <*ViewFields*> node (line 1696) and the fields to be sorted under the <*OrderBy*> node (line 1708). Here, we sort the data by customer name then by year.

```
<Views>
  <View BaseViewID="1" Type="HTML" WebPartZoneID="Main" DisplayName="ExcelWriterView" DefaultView="TRUE" ImageUr
  <View Name="WordWriterView" Type="HTML" DisplayName="WordWriterView" Url="WordWriterView.aspx" BaseViewID="2"
    <ViewFields>
      <FieldRef Name="orderyear" />
      <FieldRef Name="orderdate" />
      <FieldRef Name="orderid" />
      <FieldRef Name="orderamount" />
      <FieldRef Name="customer" />
      <FieldRef Name="address" />
      <FieldRef Name="city" />
      <FieldRef Name="state" />
      <FieldRef Name="zipcode" />
    </ViewFields>
    <Query>
      <OrderBy>
        <FieldRef Name="customer" />
        <FieldRef Name="orderyear" />
      </OrderBy>
    </Query>
    <RowLimit Paged="TRUE">1000</RowLimit>
```

Step 3: Provision custom list instance

The Elements.xml file in the *DataListInstance* folder provisions an instance, *SoftArtisans Data*, of the custom list defined above. The instance contains 200 fictitious orders, which are specified under the *<Data>* node.

```
<ListInstance
    TemplateType="999"  —— must match Type in list definition
    Id="SoftArtisans.Tutorial.DataListInstance"
    Url="SoftArtisans.Tutorial.DataListInstance"
    Title="SoftArtisans Data"
    Description="Sample data for use by SoftArtisans OfficeWriter"
    OnQuickLaunch="true">
  <Data>
    <Rows>
      <Row>
        <Field Name="orderid">1165</Field>
        <Field Name="orderdate">2010-03-11T00:00:00</Field>
        <Field Name="orderamount">8380</Field>
        <Field Name="customer">Chelsea H. Rasmussen</Field>
        <Field Name="address">864-662 Lacus. Rd.</Field>
        <Field Name="city">Los Angeles</Field>
        <Field Name="state">California</Field>
        <Field Name="zipcode">70842</Field>
      </Row>
```

Step 4: Provision template document library

The Elements.xml file in the *TemplateLibInstance* folder provisions a document library, *SoftArtisans Templates*, as well as a copy of the Word template. The *FeatureId* attribute of the instance must match the ID of the SharePoint DocumentLibrary feature.

```
<ListInstance                                          must match ID of DocumentLibrary feature
  FeatureId="00BFEA71-E717-4E80-AA17-D0C71B360101"
  TemplateType="101"
  Id="SoftArtisans.Tutorial.Templates"
  Url="SoftArtisans.Tutorial.Templates"
  Title="SoftArtisans Templates"
  Description="Document library containing templates for use by SoftArtisans OfficeWriter"
  OnQuickLaunch="true" />
<Module Name="Templates" List="101" Path="Templates" Url="SoftArtisans.Tutorial.Templates">
  <File Url="template.xlsx" Type="GhostableInLibrary" />
  <File Url="template.docx" Type="GhostableInLibrary" />
</Module>
```

Step 5: Provision output document library

In the *OutputLibInstance* folder, we provision an empty document library, *SoftArtisans Output*, where the document generated by WordWriter will be saved.

```
<ListInstance
  FeatureId="00BFEA71-E717-4E80-AA17-D0C71B360101"
  TemplateType="101"
  Id="SoftArtisans.Tutorial.Output"
  Url="SoftArtisans.Tutorial.Output"
  Title="SoftArtisans Output"
  Description="Document library containing documents and spreadsheets populated by SoftArtisans OfficeWriter"
  OnQuickLaunch="true" />
```

Step 6: Create custom menu action

In the *Menu* folder, we create a custom action that targets an application page which we create in Step 7. For simplicity, we specify that the action can be attached to any list rather than to a specific list instance. The ID of the list and the URL of the site are passed to the application page as a query string.

```
<CustomAction
Id="SoftArtisans.Tutorial.MenuItemWord"
GroupId="ActionsMenu"
Location="Microsoft.SharePoint.StandardMenu"
RegistrationType="List"
Sequence="1000"
Title="Export to SoftArtisans OfficeWriter for Word"
ImageUrl="/_layouts/images/softartisanstutorial/actionicon_word.gif">
  <UrlAction Url="~site/_layouts/SoftArtisansTutorial/WordWriter.aspx?ListId={ListId}"/>
</CustomAction>
```

Step 7: Create application page

The application page uses WordTemplate to export the items in the custom list as a Word document. Using the SharePoint integration extension methods, we can do so with just a few lines of code. Furthermore, the SharePoint-specific methods parallel existing methods, allowing WordWriter to behave in a consistent and expected manner.

The output should consist of form letters, one per customer, listing all orders, grouped by year, made by that customer. The steps to do so using the SharePoint integration feature for WordTemplate are as follows:

1. Reference the WordWriter assembly, SoftArtisans.OfficeWriter.WordWriter.dll
2. Reference the WordWriter SharePoint integration assembly, SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll
3. Reference Microsoft.SharePoint.dll, which can be found in the *ISAPI* folder of the 12 Hive
4. Import the SoftArtisans.OfficeWriter.WordWriter and Microsoft.SharePoint namespaces
5. Instantiate WordTemplate:

   ```
   WordTemplate wt = new WordTemplate();
   ```

6. Load the template directly from the document library:

   ```
   wt.Open(docLib, "template.docx");
   ```

   where *docLib* is the SPList for the template document library.

7. Set the data source for the template:

   ```
   wt.SetRepeatBlock(list.Views["WordWriterView"], list, "MailMerge");
   ```

   The SetRepeatBlock extension method takes an SPList as well as a specific SPView. The view allows us to refine the items retrieved from the list. Within the view we can select which columns to return, which columns to sort on, the total number of items, and more. Here we use a custom view specified in the custom list definition, *WordWriterView*, which sorts the orders by customer name then by year. The last parameter, *MailMerge*, is the name of the outermost repeat block.

8. Call the WordTemplate.Process() method to allow WordWriter to populate the template:

```
wt.Process();
```

9. Save the populated document to another document library, overwriting an existing file with the same name if necessary:

```
bool currentAllowUnsafeUpdates = outputLib.ParentWeb.AllowUnsafeUpdates;
outputLib.ParentWeb.AllowUnsafeUpdates = true;

wt.Save(outputLib, "populated.docx", true);

outputLib.Update();
outputLib.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates;
```

   where *outputLib* is the SPList of the output document library. Because the custom action invokes the application page with a GET request, we must temporarily enable *AllowUnsafeUpdates* before saving the file. It's not necessary to enable *AllowUnsafeUpdates* when updating a SharePoint site for an application page invoked from a POST request or for a desktop application.

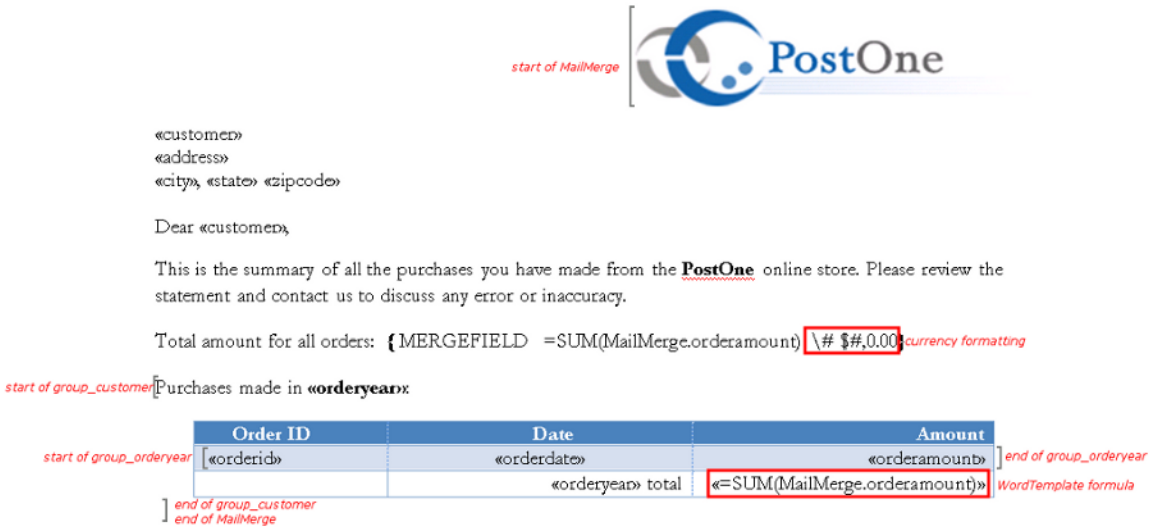
10. Instead of or in addition to saving the file to a document library, you can use other overloads of the WordTemplate.Save method. For example, to stream a copy of the document to the browser:

```
wt.Save(Response, "populated.docx", false);
```

Step 8: Create the template

Follow the documentation pages for creating a regular template or grouping template.

Here, we use the below grouping template.

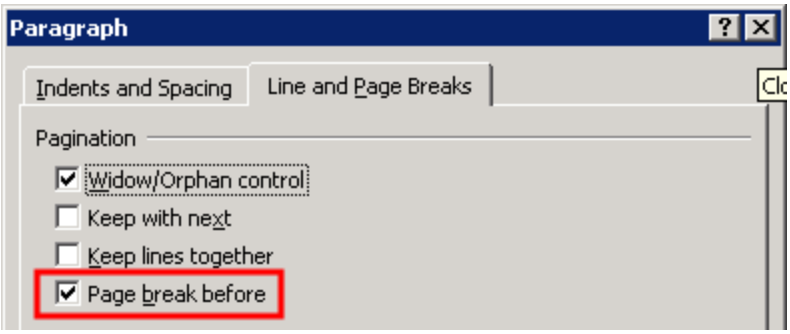


A bookmark is delimited by square brackets, which are not printed. In order to show the bookmark in Word 2007, select *Office menu* > *Word Options* > *Advanced* > *Show document content*, then check *Show bookmarks*.

In order to group by customer name, we insert the bookmark named *group_customer*, where *customer* is the name of the desired column. Then in order to group each customer's orders by year, we inserted a nested bookmark named *group_orderyear*.

The grouping bookmarks are enclosed in an outermost bookmark named *MailMerge*, which is passed to the WordTemplate.SetRepeatblock method. So that each letter begins on a new page, insert a page break before the beginning of each page:

1. Place the cursor at the top of the page
2. Open the Paragraph setting dialog
3. On the *Line and Page Breaks* tab, check *Page break before*



We also insert two sub-totals for order amount using WordTemplate formula. The sub-total for each order year is outside the *group_orderyear* bookmark, while the sub-total for each customer is outside the *group_customer* bookmark.

We show the field codes of the merge field for the customer sub-total to illustrate merge field formatting. To show the field codes, right-click on the merge field and select *Toggle Field Codes* or use the shortcut ALT-F9. We use U.S.-currency formatting for sub-total merge fields. We apply formatting in a similar manner to the date field.

Step 9: Managing the SharePoint solution package

The attached archive contains a SharePoint solution (.WSP) package that you can deploy immediately. Otherwise, you can build the solution manually:

1. Build the *SoftArtisansTutorial* project in Visual Studio 2008
2. At the command prompt, navigate to the *SOLUTION* folder of the tutorial project
3. Execute the *MakeCab.cmd* script (you need to have Microsoft's makecab.exe utility in the system path or in the same directory) to create the WSP package based on the *SoftArtisansTutorial.ddf* file

To deploy the SharePoint solution package:

1. At the command prompt, navigate to the *SOLUTION* folder of the tutorial project
2. Execute the *DeploySolution.cmd* script
3. Activate the *SoftArtisans Tutorial* feature for the desired site

Deployment to farm
The script deploys the SharePoint solution on the local computer only. If you want to deploy the solution to all servers in a farm, edit *DeploySolution.cmd* and replace the deployment command with the following:

```
stsadm.exe -o deploysolution -name SoftArtisansTutorial.wsp -allowGacDeployment -immediate -allcontenturls
stsadm.exe -o execadmsvcjobs
```

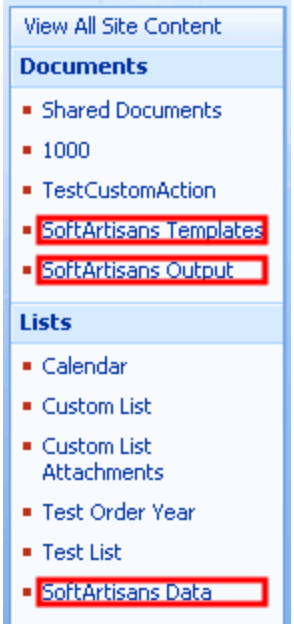
To retract the SharePoint solution package:

1. Deactivate the *SoftArtisans Tutorial* feature on the desired site
2. At the command prompt, navigate to the *SOLUTION* folder of the tutorial project
3. Execute the *RetractSolution.cmd* script

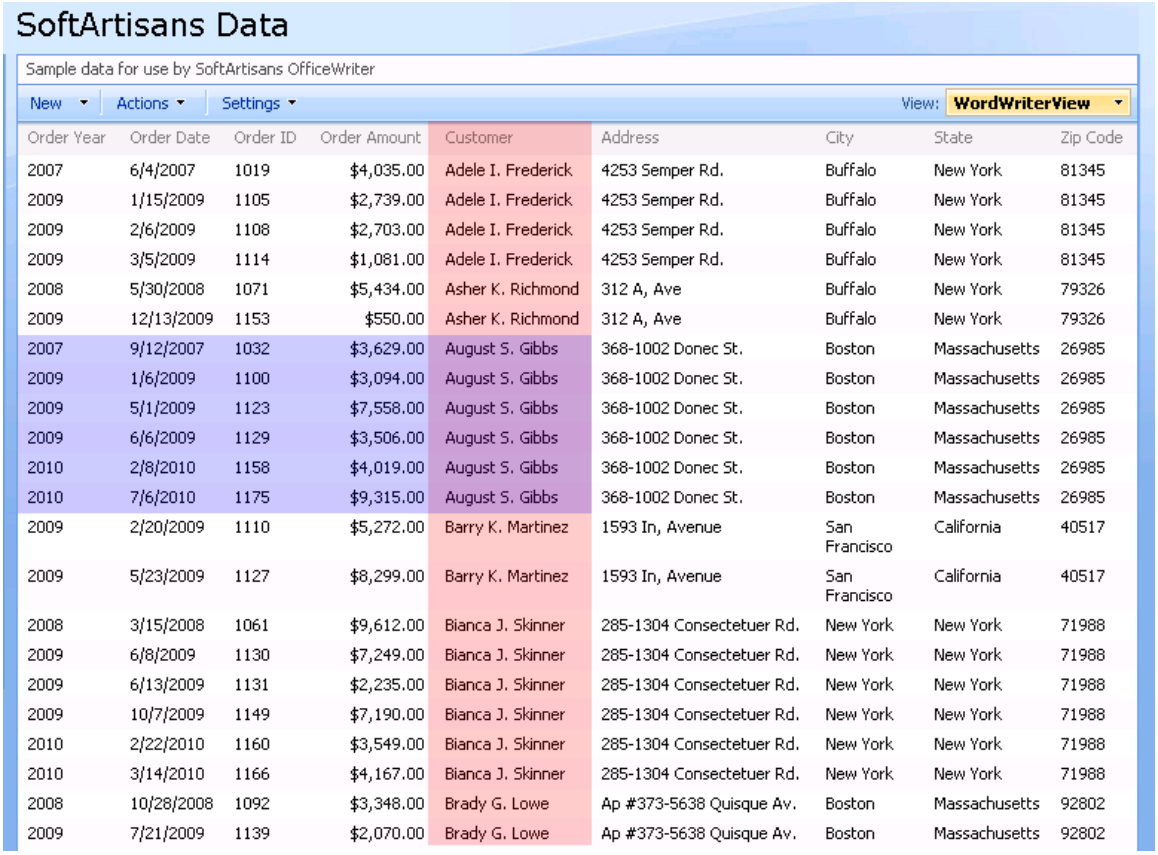
Step 10: Using the tutorial

Activating the *SoftArtisans Tutorial* feature automatically creates the following lists, which should be shown on the Quick Launch bar:

- The *SoftArtisans Templates* document library, which contains *template.docx*
- The empty *SoftArtisans Output* document library
- The *SoftArtisans Data* custom list, which contains sample data



Open the *SoftArtisans Data* list. Select *WordWriterView* from the View drop-down. Note that the data is sorted by customer, then by order year.



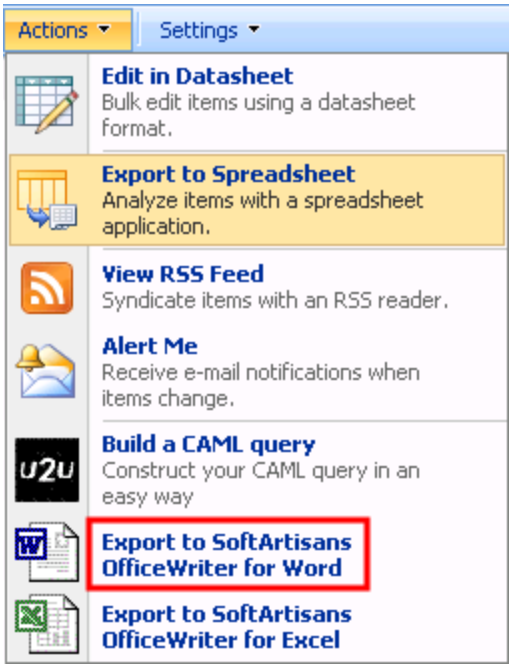
SoftArtisans Data

Sample data for use by SoftArtisans OfficeWriter

New | Actions | Settings | View: WordWriterView

Order Year	Order Date	Order ID	Order Amount	Customer	Address	City	State	Zip Code
2007	6/4/2007	1019	$4,035.00	Adele I. Frederick	4253 Semper Rd.	Buffalo	New York	81345
2009	1/15/2009	1105	$2,739.00	Adele I. Frederick	4253 Semper Rd.	Buffalo	New York	81345
2009	2/6/2009	1108	$2,703.00	Adele I. Frederick	4253 Semper Rd.	Buffalo	New York	81345
2009	3/5/2009	1114	$1,081.00	Adele I. Frederick	4253 Semper Rd.	Buffalo	New York	81345
2008	5/30/2008	1071	$5,434.00	Asher K. Richmond	312 A, Ave	Buffalo	New York	79326
2009	12/13/2009	1153	$550.00	Asher K. Richmond	312 A, Ave	Buffalo	New York	79326
2007	9/12/2007	1032	$3,629.00	August S. Gibbs	368-1002 Donec St.	Boston	Massachusetts	26985
2009	1/6/2009	1100	$3,094.00	August S. Gibbs	368-1002 Donec St.	Boston	Massachusetts	26985
2009	5/1/2009	1123	$7,558.00	August S. Gibbs	368-1002 Donec St.	Boston	Massachusetts	26985
2009	6/6/2009	1129	$3,506.00	August S. Gibbs	368-1002 Donec St.	Boston	Massachusetts	26985
2010	2/8/2010	1158	$4,019.00	August S. Gibbs	368-1002 Donec St.	Boston	Massachusetts	26985
2010	7/6/2010	1175	$9,315.00	August S. Gibbs	368-1002 Donec St.	Boston	Massachusetts	26985
2009	2/20/2009	1110	$5,272.00	Barry K. Martinez	1593 In, Avenue	San Francisco	California	40517
2009	5/23/2009	1127	$8,299.00	Barry K. Martinez	1593 In, Avenue	San Francisco	California	40517
2008	3/15/2008	1061	$9,612.00	Bianca J. Skinner	285-1304 Consectetuer Rd.	New York	New York	71988
2009	6/8/2009	1130	$7,249.00	Bianca J. Skinner	285-1304 Consectetuer Rd.	New York	New York	71988
2009	6/13/2009	1131	$2,235.00	Bianca J. Skinner	285-1304 Consectetuer Rd.	New York	New York	71988
2009	10/7/2009	1149	$7,190.00	Bianca J. Skinner	285-1304 Consectetuer Rd.	New York	New York	71988
2010	2/22/2010	1160	$3,549.00	Bianca J. Skinner	285-1304 Consectetuer Rd.	New York	New York	71988
2010	3/14/2010	1166	$4,167.00	Bianca J. Skinner	285-1304 Consectetuer Rd.	New York	New York	71988
2008	10/28/2008	1092	$3,348.00	Brady G. Lowe	Ap #373-5638 Quisque Av.	Boston	Massachusetts	92802
2009	7/21/2009	1139	$2,070.00	Brady G. Lowe	Ap #373-5638 Quisque Av.	Boston	Massachusetts	92802

Open the Action drop-down menu. Select *Export to SoftArtisans OfficeWriter for Word*.



WordWriter then retrieves the template, pulls the data from the list, and populates the template. The generated document is saved to the *SoftArtisans Output* document library and streamed to the browser. Below is the letter for one customer.

August S. Gibbs
368-1002 Donec St.
Boston, Massachusetts 26985

Dear August S. Gibbs,

This is the summary of all the purchases you have made from the **PostOne** online store. Please review the statement and contact us to discuss any error or inaccuracy.

Total amount for all orders: $31,121.00

Purchases made in **2007**:

Order ID	Date	Amount
1032	September 12, 2007	$3,629.00
	2007 total	$3,629.00

Purchases made in **2009**:

Order ID	Date	Amount
1100	January 6, 2009	$3,094.00
1123	May 1, 2009	$7,558.00
1129	June 6, 2009	$3,506.00
	2009 total	$14,158.00

Purchases made in **2010**:

Order ID	Date	Amount
1158	February 8, 2010	$4,019.00
1175	July 6, 2010	$9,315.00
	2010 total	$13,334.00

SharePoint Solution - Word Export Plus

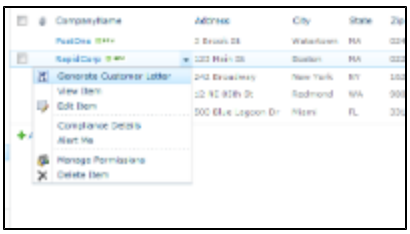
Table of Contents

- Word Export Plus Introduction
  - How it Works
  - Fully Customizable By Developers
- Installation and Configuration
  - Installation
    - Prerequisites and System Requirements
    - Installation
  - Configuration
    - Feature Activation
    - Defining Export Settings
- Doing More with OfficeWriter

Word Export Plus Introduction

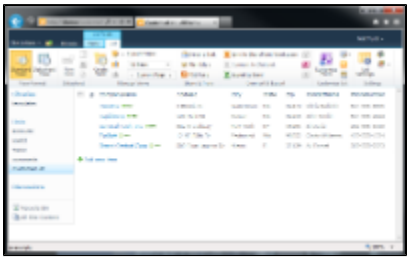
Word Export Plus is a SharePoint solution that demonstrates a usage of the OfficeWriter API in SharePoint. This solution adds a new context menu button to list items, allowing you to export the list data to a pre-formatted Word template that can be designed yourself in Word, or automatically generated by Word Export Plus.

The "Generate Customer Letter" menu item below is an example of a Word Export Plus button. You can customize the button text based on your requirements.



How it Works

Word Export Plus uses the OfficeWriter API under SharePoint's hood to merge SharePoint list item data into a Word document. For example, say you have a list like this one, with CompanyName, Address, City, State, Zip, ContactName, and PhoneNumber columns:



You could design a template like the one shown below in Word. Put anything you want into the Word document, and insert merge fields that correspond to the names of the columns in the list.

When you choose to export the list item from the context menu, Word Export Plus pulls the values from the list item you selected, and then uses OfficeWriter to open the template file, replace the merge fields with actual data, and stream it to the browser:

Fully Customizable By Developers

You can install OfficeWriter and Word Export Plus and use it as-is, customize the code to fit your own needs, or simply use it as an educational tool to better understand the capabilities of OfficeWriter.

Installation and Configuration

Installation

The first step is to install OfficeWriter and Word Export Plus on the SharePoint web farm.

Prerequisites and System Requirements

The following is a list of prerequisites for this solution installer.

To run the solution installer:

1. **OfficeWriter Enterprise key** - An Enterprise key is required to use an OfficeWriter SharePoint solution. If you do not have an enterprise license key, you can obtain a free 30-day evaluation by visiting http://www.officewriter.com or by contacting SoftArtisans at contact@softartisans.com.
2. **SharePoint 2010** - This solution is supported only on SharePoint 2010
3. **Local machine administrative privileges** - The installer must be run with local administrative privileges
4. **Farm administrator privileges** - The installer requires farm administrator privileges to run

Installation

All installation steps are assumed to occur on the machine where the SharePoint web app resides.

1. **Install OfficeWriter** - If you haven't already, install the OfficeWriter product. If you don't have the OfficeWriter product already installed, you can download an evaluation copy by visiting http://www.officewriter.com or by contacting SoftArtisans at contact@softartisans.com.
2. **Install the solution on the farm** - Run the Word Export Plus installer. This will install the solution into the SharePoint farm.
3. **Deploy the solution** - Once the solution is on the SharePoint farm, deploy it by following these steps:
    a. Navigate to "System Settings" > "Farm Management" > "Manage Farm Solutions"
    b. Click "Word Export Plus.wsp"
    c. Click "Deploy Solution"
    d. Specify a deployment time
    e. Click "OK"

Configuration

Feature Activation

Once the Installation steps are complete, the solution will appear as a SharePoint Feature that can be activated and managed by a site administrator. Follow these steps to configure Word Export Plus:

1. Browse to the SharePoint site in which you want to use this solution.
2. Under the "Site Actions" menu, select "Site Settings."
3. Select "Manage site features" in the "Site Actions" section.
4. Locate the "OfficeWriter Solution Gallery - Word Export Plus" item, and click "Activate." When it has been activated, it will look like this:

Defining Export Settings

In this step, you will enable the Word Export Plus button on a specific list item, select or create a template file that it will export the item's content to, and define the text for the context menu item.

1. From "Site Settings", click "OfficeWriter Solution: Word Export Plus" in the "Look and Feel" section. This will bring up a configuration screen.
2. In the "SharePoint List" section, click "Browse…" and choose a list in your site on which you want to activate Word Export Plus.
3. In the "Word Template" section, select "Create a new template file", and then choose the columns from the list that you want to appear in the export. Word Export Plus will automatically create a document for you with the merge fields that you choose (you can go back later and customize the template in Word). You can also select a previously created template that you've uploaded to any library in the SharePoint site.
4. If creating a new template, specify a document library in which you want to hold the new template.
5. Type a name for the new template file
6. Finally, in the "Action Name" section specify the text you want to appear on the context menu item, and provide a (optional) description.

Here is an example of the configuration screen for the sample list described above:

Doing More with OfficeWriter

Now that you have your list associated with a template file in the document library, you can edit the template and add more formatting, content, or merge fields that correspond to column names in the list.

For more information about template files, see here:
http://wiki.softartisans.com/display/WW40/Creating+a+Template

WordWriter in ASP

- How to Use WordWriter in ASP
- ASP Reference
  - WordTemplate Object
  - BookmarkFieldMarkers Method
  - Bookmarks Property
  - ContentType Property
  - CultureInfo Property
  - FieldMarkers Property
  - LicenseKey Property
  - Open Method
  - Process Method
  - Save Method
  - SetDataSource Method
  - SetRepeatBlock Method
  - Version Property

How to Use WordWriter in ASP

> The WordWriter COM-callable wrapper is no longer included in the standard install for OfficeWriter. If you need WordWriter COM, please contact SoftArtisans directly.

A WordWriter COM callable wrapper (CCW) that allows you to use the .NET WordTemplate object from ASP is available upon request. To use WordWriter in ASP:

1. All of the system requirements for running WordWriter in ASP.NET must be satisfied.

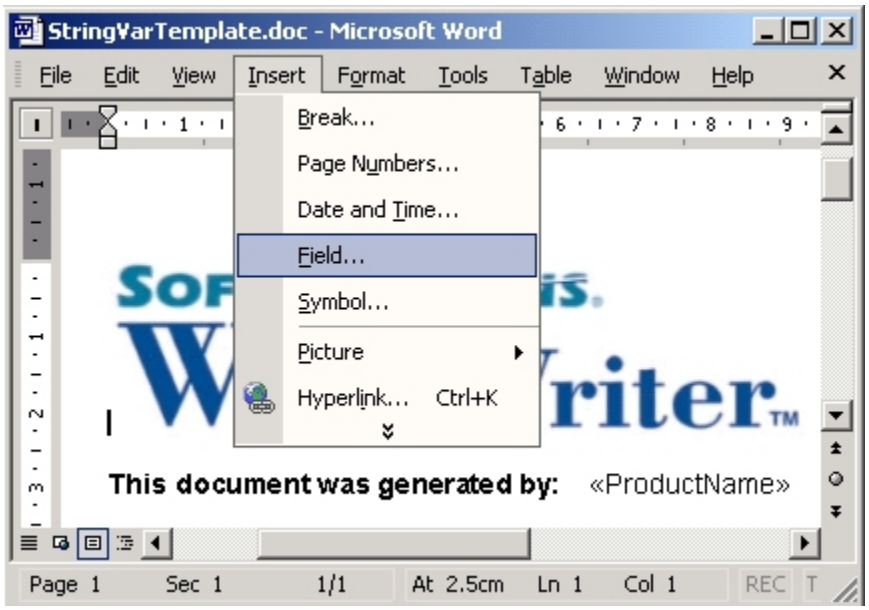
2. SoftArtisans.OfficeWriter.WordWriter.dll must be installed in the Global Assembly Cache. For instructions, see ASP Installation .

Example: Generating a Word File in ASP

The following ASP example generates a file from a template that contains a single merge field. The data source is a single-element array.

To generate a Word file with WordWriter, first create a template:

1. In Microsoft Word, create a new file.
2. From the **Insert** menu, select **Field...** to open the Field dialog.



3. From the **Field names** list, select **MergeField**.
4. In the **Field name** text box, enter a name for the merge field. For example, enter *ProductName*.
5. Click Ok. The merge field will be displayed in the document in the format <<Field Name>>.
6. To insert additional merge fields, repeat steps 2-4.

Next, create an ASP script that uses WordWriter to open the template, fill in merge field values, and generate a new Word file. The following VBScript example opens a template that contains a single merge field - **<<ProductName>>**. The data source for the merge field will be a single-element array.

To generate a new Word file with WordWriter in ASP:

1. Create an instance of the WordTemplate object:

```
Dim wwt
Set wwt = Server.CreateObject("SoftArtisans.WordTemplate")
```

2. Call WordTemplate.Open to open a template Word file:

```
wwt.Open("C:\Templates\Template.doc")
```

   The Open method takes the complete path and file name of the template Word file. Note that in ASP you cannot pass a stream to Open.

   > WordWriter supports Microsoft Word 97, 2000, 2002 (XP), 2003, and 2007. Do not use Open to open files created in earlier versions of Microsoft Word.

3. Call WordTemplate.SetDataSource to bind the template's merge fields to a data source:

```
Dim arrValue(0)
Dim arrName(0)
arrValue(0) = "SoftArtisans WordWriter"
arrName(0) = "ProductName"
wwt.SetDataSource arrValue, arrName
```

To set a data source to a single value, put the value in a single-element array, and the merge field name in a single-element array. Pass the two arrays to WordTemplate.SetDataSource.

4. Call WordTemplate.Process to enter values from the data source into the template's merge fields:

```
wwt.Process
```

5. Call WordTemplate.Save to save the generated file or stream it to the browser. The following streams the file to the browser:

```
wwt.Save Response, "StringVarOutput.doc", False
```

When you pass the ASP Response object to Save, WordWriter will stream the file to the client. The method's second parameter specifies a default name for the generated file; this name will be displayed in the browser's File Download dialog. The third parameter specifies whether the file should open in the browser window or in Microsoft Word; if the parameter is set to False, the file will open in Word.

ASP Reference

WordTemplate Object	Represents a WordWriter template. A WordWriter template is a file created in Microsoft Word that contains merge fields.
BookmarkFieldMarkers Method	Returns the names of all merge fields in the specified bookmark.
Bookmarks Property	Returns the names of all bookmarks in a WordWriter template.
ContentType Property	Sets or returns the content-type header that will be sent to the browser with the generated Word document.
CultureInfo Property	The CultureInfo property allows you to override the server's default locale when generating a new Word file.
FieldMarkers Property	Returns the names of all merge fields in the main document.
LicenseKey Property	Returns the registered 23-character license key that enables WordWriter.
Open Method	Opens a WordWriter template.
Process Method	Enters data source values in a template's merge fields, and creates an image of the output file (the new document) in memory.
Save Method	Saves the generated Word file on the server or streams it to the client.
SetDataSource Method	Sets the main document data source to a one-dimensional array or an ADO recordset.
SetRepeatBlock Method	Sets the specified repeat block's data source to an ADO RecordSet or an array.
Version Property	Returns the exact version of WordWriter.

WordTemplate Object

The WordTemplate object represents a WordWriter template. A WordWriter template is a file created in Microsoft Word that contains merge fields.

To create an instance of the ExcelTemplate object in ASP, use:

```
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")
```

WordTemplate Properties	WordTemplate Methods
• Bookmarks • ContentType • CultureInfo • FieldMarkers • LicenseKey • Version	• BookmarkFieldMarkers • Open • Process • Save • SetDataSource • SetRepeatBlock

BookmarkFieldMarkers Method

Description

Returns the names of all merge fields in the specified bookmark.

Signature: VBScript

```
Public Function BookmarkFieldMarkers(aBookmark As String) As String()
```

Parameters

aBookmark

Returns

The names of all merge fields in the specified bookmark.

Remarks

Bookmarks are used to define repeat blocks. A repeat block is a fragment in the template document that contains merge fields and that will be repeated for each row in a data source.

Examples

```
'-- Create the WordTemplate object.
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")

'-- Open a template
oWW.Open("C:\TemplateDirectory\Template.doc")

'-- Get all the bookmark names in the document.
allBookmarks = oWW.Bookmarks

'-- Get all the merge fields in the first bookmark.
someFields = oWW.BookmarkFieldMarkers(allBookmarks(0))
```

Bookmarks Property

Description

Returns the names of all bookmarks in a WordWriter template.

Signature: VBScript

```
Public Property Bookmarks As String()
```

Remarks

Bookmarks are used to define repeat blocks. A repeat block is a fragment in the template document that contains merge fields and that will be repeated for each row in a data source.

Example

```
'--- Create the WordTemplate object.
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")

'--- Open a template.
oWW.Open("C:\TemplateDirectory\Template.doc")

'--- Get all the bookmark names in the document.
allBookmarks = oWW.Bookmarks
```

ContentType Property

Description

Sets or returns the content-type header sent to the browser with the generated Word document.

Signature: VBScript

```
Public Property ContentType As String
```

Example

```
Set WordTempl = Server.CreateObject("SoftArtisans.WordTemplate")
WordTempl.Open Server.MapPath("./BasicTemplate.doc")

WordTempl.ContentType = "application/vnd.ms-word"

WordTempl.SetDataSource ValuesArr, NamesArr
WordTempl.Process
WordTempl.Save Response, "BasicGenerated.doc", False
```

CultureInfo Property

Description

Sets the locale to a value other than the server's default locale when generating a new Word file

Signature: VBScript

```
Public WriteOnly Property CultureInfo As Variant
```

Remarks

The CultureInfo property allows you to override the server's default locale when generating a new Word file. For example, to generate a US English document on a French operating system, you could use SetCultureInfo to set the Word file's locale to US English ("en-US"). For a list of culture name strings, see CultureInfo Class.

Example

```
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")

'--- Set the Word file's locale to US English.
oWW.CultureInfo = "en-US"
oWW.Open(Server.MapPath("./Template.doc"))
oWW.SetDataSource arrValues, arrNames
oWW.Process
oWW.Save Response, "Output.doc", False
Set oWW = Nothing
```

FieldMarkers Property

Description

Returns the names of all merge fields in the main document.

Signature: VBScript

```
Public ReadOnly Property FieldMarkers As String()
```

Remarks

The main document is any part of the template that is not within a repeat block. (A repeat block is a fragment in the template document - defined by a Word bookmark - that contains merge fields and that will be repeated for each row in a data source.)

Example

```
'--- Create the WordTemplate object.
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")

'--- Open a template.
oWW.Open("C:\TemplateDirectory\Template.doc")

'--- Get all the merge fields in the main document.
allMergefields = oWW.FieldMarkers
```

Markers

LicenseKey Property

Description

Returns the registered 23-character license key that enables WordWriter.

Signature: VBScript

```
Public Property LicenseKey As String
```

Remarks

The key contains information about WordWriter, including product name, platform, and version. Only the 23-character key is stored in the registry. To determine which licensed version is currently installed on the server, run LicenseManager.exe (in OfficeWriter\WordWriter) and enter the license key retrieved by the LicenseKey property.

Example

```
Set WordTempl = Server.CreateObject("SoftArtisans.WordTemplate")
WordTempl.Open Server.MapPath("./BasicTemplate.doc")

key = WordTempl.LicenseKey
NamesArr = Array("LicenseKey", "DateTime")
ValuesArr = Array(Key, Now())

WordTempl.SetDataSource ValuesArr, NamesArr
WordTempl.Process
Response.Clear
WordTempl.Save Response, "BasicGenerated.doc", False
```

Open Method

Description

Opens the WordWriter template specified by the templateFileName parameter.

Signature: VBScript

```
Public Sub Open(templateFileName As String)
```

Parameters

templateFileName

The complete path and file name of the template Word file.

Remarks

A WordWriter template is a file created in Microsoft Word that contains merge fields.

WordWriter supports Microsoft Word 97, 2000, 2002 (XP), 2003, and 2007. Do not use Open to open files created in earlier versions of Microsoft Word.

> ⚠ WordWriter templates can be created in Windows 2003. However, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.

Examples

```
arrValues = Array("SoftArtisans", "WordWriter", _
     "http://www.softartisans.com")
arrNames = Array("CompanyName", "ProductName", "URL")
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")
oWW.Open("C:\TemplateDirectory\Template.doc")
oWW.SetDataSource arrValues, arrNames
oWW.Process
oWW.Save Response, "Output.doc", False
Set oWW = Nothing
```

Process Method

Description

The Process method enters data source values in a template's merge fields, and creates an image of the output file (the new document) in memory. The save method can then save the output file to disk, stream it to the browser, or both.

Signature: VBScript

```
Public Sub Process()
```

Remarks

Call Process after SetDataSource and before Save.

You can call Process more than once for a single instance of WordTemplate. This allows you to generate multiple files from a single template and an updated data source.

Examples

```
arrValues = Array("SoftArtisans", "WordWriter", _
     "http://www.softartisans.com")
arrNames = Array("CompanyName", "ProductName", "URL")
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")
oWW.Open(Server.MapPath("./Template.doc"))
oWW.SetDataSource arrValues, arrNames

'--- Call WordTemplate.Process to populate the
'--- template with the new data.
oWW.Process
oWW.Save Response, "Output.doc", False
Set oWW = Nothing
```

Save Method

Description

Saves the generated Word file on the server or streams it to the client.

Signature: VBScript

```
Public Sub Save(Output As Variant, optional attachmentName As String, optional OpenInBrowser As
Boolean)
```

Parameters

OutputOutput

Either a file path or the ASP Response object. To save the generated Word file on the server, Output should be a complete path and file name. WordWriter will save the file to this location. If a file with the same name exists, it will be overwritten by the new Word file. To stream the generated Word file to the client,Output should be the Response object.

attachmentName

When the generated file is streamed to the client, you can set this optional parameter which specifies a name for the generated Word file; this name will be displayed in the download dialog when the file is streamed to the browser.

openInBrowser

When the generated file is streamed to the client, you can set this optional parameter. If openInBrowser is set to true, and the user chooses to open the file, the file will open in the browser window. If openInBrowser is set to false, and the user chooses to open the file, the file will open in Microsoft Word. By default, the file will open in the browser window.

> ⚠ Not all browsers can embed a Word file in the browser window.

Remarks

The first parameter of the Save method may be either a file path or the ASP Response object. If you pass Save a file path, WordWriter will save the generated file on the server. If you pass Save the Response object, WordWriter will stream the generated file to the client.

You can call Save more than once for a single instance of WordTemplate. This allows you to save more than one copy of a generated file, and/or both save the file on the server and stream it to the client.

Examples

```
'--- Save the generated file on the server.
arrValues = Array("SoftArtisans", "WordWriter", "http://www.softartisans.com")
arrNames = Array("CompanyName", "ProductName", "URL")
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")
oWW.Open("C:\TemplateDirectory\Template.doc")
oWW.SetDataSource arrValues, arrNames
oWW.Process
oWW.Save "C:\Output\Output.doc"
Set oWW = Nothing
```

```
VBScript
'--- Stream the generated file to the client.
arrValues = Array("SoftArtisans", "WordWriter", "http://www.softartisans.com")
arrNames = Array("CompanyName", "ProductName", "URL")
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")
oWW.Open("C:\TemplateDirectory\Template.doc")
oWW.SetDataSource arrValues, arrNames
oWW.Process
oWW.Save Response, "Output.doc", False
Set oWW = Nothing
```

SetDataSource Method

Description

Sets a template's data source to a one-dimensional array or an ADO recordset.

Signature: VBScript

```
Public Sub SetDataSource(DataSource As Variant, optional arrNames As Variant, optional dataSetName As
Variant)
```

Parameters

DataSource

This parameter specifies the data source, which may be either a 1-dimensional array or a recordset. If the data source is a multiple row recordset, WordWriter will use the first row of the recordset as the data source.

arrNames

Pass this optional parameter to SetDataSource, if DataSource is an array. arrNames is an array of data source field names. These must be the same as the corresponding merge field names in the template.

dataSetName

Pass this optional parameter to SetDataSource, if this template uses multiple DataSources.dataSetName is a string corresponding to the name of the data source on the template file.

Remarks

SetDataSource sets a data source for the main document(the part of the document that is not within a repeat block). To set additional data sources, create repeat blocks in the template and call SetRepeatBlock.

The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

If the data source assigned by SetDataSource is an array, you must pass the method the second optional parameter arrNames. arrNamesis an array of field names. The two arrays (values and field names) arrays must contain the same number of elements. Each name in the array of field names must be the same as the corresponding merge field name in the template.

Examples

```
'--- Set a 1-dimensional array data source.

'--- Create an array of values. These values will be
'--- inserted in the template's merge fields.
arrValues = Array("SoftArtisans", "WordWriter", _
     "http://www.softartisans.com")

'--- Create an array of field names. These must be the
'--- same as the corresponding merge field names in the template.
'--- The array of values and the array of names must contain
'--- the same number of elements.
arrNames = Array("CompanyName", "ProductName", "URL")

Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")
oWW.Open(Server.MapPath("./Template.doc"))

'--- Set the data source of the document by passing
'--- the value array and the name array that we initialized above.
oWW.SetDataSource arrValues, arrNames

oWW.Process
oWW.Save Response, "ArrayVarsOutput.doc", False
```

```
'--- Set the data source to an ADO recordset.
strConnString = Application("connString")
strSQL = "SELECT TOP 1 TitleOfCourtesy, FirstName, LastName," & _
     Title FROM Employees"
Set oConn = Server.CreateObject("ADODB.Connection")
oConn.Open strConnString
Set oRS = oConn.Execute(strSQL)
Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")
oWW.Open Server.MapPath("./Template.doc")

'--- Pass the RecordSet object to SetDataSource().
oWW.SetDataSource oRS

oWW.Process
oWW.Save Response, "DataTableOutput.doc", False
```

SetRepeatBlock Method

Description

Sets the specified repeat block's data source to an ADO RecordSet or an array.

Signature: VBScript

```
Public Sub SetRepeatBlock(data As Variant, bookmark As String, optional column_names As Variant,
optional maxRows as Long)
```

Parameters

data

The array or RecordSet to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to SetRepeatBlock must exist in the template Word file. To see a list of template bookmark names:
1. Open the template in Microsoft Word.
2. Open the Edit menu.

3. Select Go To...
4. Select Bookmark.
5. Under Enter bookmark name, open the drop-down list of bookmark names.

column_names

A String array of data source field names. Pass column_names to SetRepeatBlock if the data source is an array. The column names must be the same as the corresponding merge field names in the template.

maxRows

Specifies the maximum number of rows to import from the data source. If maxRows is not specified, WordWriter will import all rows in the data source.

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter bookmarkmust exist in the template.

If you pass an empty data source to SetRepeatBlock, WordWriter will not include the specified repeat block in the generated Word file.

Examples

```
'--- Get a RecordSet of recipient information.
Set RecSet = GetRecipientData()

Set oWW = Server.CreateObject("SoftArtisans.WordTemplate")
oWW.Open Server.MapPath("./template.doc")

'--- Set a data source for the repeat block "Page"
'--- "RecSet" is the RecordSet containing orders
'--- recipient information.
oWW.SetRepeatBlock RecSet, "Page"

oWW.Process
oWW.Save Response, "FormLetterOutput.doc", False

Function GetRecipientData()

     Dim SQL
     Dim Conn

     '--- Query the database for the names and addresses of recipients
     SQL = "SELECT FirstName + ' ' + LastName As RecipientName, " & _
          "Address As RecipientStreetAddr, " & _
          "City As RecipientCity, " & _
          "Region As RecipientState, " & _
          "PostalCode As RecipientZip FROM Employees"

     '--- Fill a RecordSet with recipient info
     Set Conn = Server.CreateObject("ADODB.Connection")
     Conn.Open Application("connstring")

     Set GetRecipientData = Conn.Execute(SQL)

End Function
```

Version Property

Description

Returns the exact version of WordWriter.

Signature: VBScript

```
Public ReadOnly Property Version As String
```

Remarks

The format of the version string returned is:

```
SoftArtisans.ProductName Major.Minor.Patch.Build
(mmddyyyy-hhmmss-buildMachineId)*
```

Place Holder	Description
ProductName	A product name like "SoftArtisans.WordWriter." The product name may be different depending on what license key is in effect.
Major	Major release number.
Minor	Minor release number.
Patch	Patch or bug fix number.
Build	Build number.
mmddyyyy	Date the build was compiled.
hhmmss	Time the build was compiled.
buildMachineId	Unique computer ID for the computer on which this product was compiled.

Example

```
Set WordTempl = Server.CreateObject("SoftArtisans.WordTemplate")
WordTempl.Open Server.MapPath("./BasicTemplate.doc")

Version = WordTempl.Version
NamesArr = Array("WordWriterVersion", "DateTime")
ValuesArr = Array(Version, Now())

WordTempl.SetDataSource ValuesArr, NamesArr
WordTempl.Process
Response.Clear
WordTempl.Save Response, "BasicGenerated.doc", False
```

Programmer's Reference

WordWriter provides two development solutions that can be used separately or together:

- WordTemplate

  Introduction to WordTemplate: Create a Document with WordTemplate

  WordTemplate is the main class for template-driven document generation. This object opens a WordWriter template file, populates it with data from a specified data source, and generates a new Word document.

- WordApplication

  WordApplication is not available in WordWriterSE.

  Introduction to WordApplication: Create a Document with WordApplication

WordApplication is the main class for pure code-based document generation. This class is an engine used to open, create, and write (save or stream to a browser) documents. A single instance of WordApplication can generate multiple Word documents.

- SharePointIntegration

> SharePointIntegration is not available in WordWriterSE.

SharePointIntegration is a class of extension methods that allow you to more easily use WordWriter from within SharePoint, with support for opening and saving from SharePoint List attachments and Document Libraries as well as data binding with SharePoint Objects.

WordApplication Object Hierarchy Diagram

The following is a diagram of the major WordWriter application objects. Click any object name in the diagram to open the page for that object.

Please note that the FieldContents and ListEntry trees recurse back to the Paragraph Tree, meaning that a FieldContents object or ListEntry object can contain the same objects as a Paragraph object. Similarly, the TableCell tree recurses back to the Section tree, meaning that a TableCell object can contain the same objects as can a Section object.

- Document
  - Section
    - Paragraph
      - CharacterRun
      - InlineImage
      - Field (includes Hyperlink and MergeField)
        - Field Contents
          - (Recurses back to Paragraph subtree)
    - List
      - ListEntry
        - (Recurses back to Paragraph subtree)
    - Table
      - TableCell
        - (Recurses back to Section subtree)

WordApplication Index

The following is a complete list of all objects, methods, and properties used in WordApplication. Objects are listed at the first level alphabetically. Methods and properties are listed at the second level under their corresponding object. Click any object, method, or property for more information.

> Overloaded methods are marked with an asterisk (*) and link to a list of implemented signatures in their corresponding object page. All other objects, methods, and properties link directly to their own pages.

AbsolutePositioning object

Properties
Height
HeightExact
OriginX
OriginY
Width
X

Y

AutoFormatter object

Methods
FormatCell

Bookmark object

Properties
Name

Border object

Properties
Color
HasShadow
LineWidth
Padding
Reverse
Style

CharacterRun object

Properties
Deleted
DeletedBy
DeletedTime
Font
ModifiedBy
ModifiedTime
New
PropertiesChangedBy
PropertiesChangedTime
Style

CompatibilitySettings object

Properties
AdjustTableLineHeight
AlignRowsIndependent
AllowLeading
AllowSnapInCells
AllowTableBreaks
ApplyBreakingRules
AutospaceLike95
BalanceColumns
BorderRulesLike6
ConvertMailMergeEsc
ExactOnTop
ExpandShiftReturn
FootnotesLike97
GrowTableAutoFit
JustifyLikeWP6
LeaveBackslashAlone
LineBreaksLike97
LineWrapLike6
MapPrintColor
MirrorMargins
NoSpaceForRaisedLower
NoTabForIndent
ShapesLike97
ShowBreaksInFrames
SnapPageBorder
SpaceLikeWP5
SubBasedOnSize
SuppressExtraAfter
SuppressSpaceAfterPageBreak

SuppressSpacingLike5
SuppressSpacingLikeWP
SupressTopSpacing
SwapBorders
TableRowsApart
TruncateFontHeight
UnderlineTrailingSpaces
UseAsianBreakingRules
UseLargerSmallCaps
UsePrinterMetricsUseTableRawWidth
UseTableStylesLike2002WrapTextWithPunct
WrapTrailingSpaces

DataImportProperties object

Properties
AutoFit
AutoFormatter
ColumnIndexFilter
ColumnNameFilter
MaxColumns
MaxRows
UseColumnNames

Document object

Properties
AutoformatType
ChangeTracking
CompatibilitySettings
DateCreated
DatePrinted
DateRevised
DefaultTabWidth
DocumentProperties
DocumentProtection
GutterPosition
HyphenSettings
MergeStylesFromTemplate
NumCharacters
NumPagesOnLastOpen
NumParagraphs
NumWords
PrintDataInFormFields
PrintFacingPages
SaveDataInFormField
Sections
Styles
UseWidowControl
ViewSettings
Watermark

Methods
CreateDataImportProperties
CreateFont
CreateParagraphFormatting
CreateTableFormatting
GetBookmark
GetVariable
SetVariable

DocumentProperties object

Properties
Application
Author
Comments
Company
CreationDate
Keywords
LastSavedDate
Manager

Subject
Title

Methods
Clear
GetCustomProperty
GetProperty
RestoreDefaults
SetCustomProperty
SetProperty

DocumentProtection object

Properties
AnnotationsLocked
AutoFormatOverride
DocumentProtected
EnforceDocumentProtection
SelectOnlyFormFields
StylesLocked

Element object

Properties
Children
ElementType
Length
Start
Text

Methods
CreateBookmark
CreateSectionAfter
CreateSectionBefore
DeleteElement
GetElements
GetPosition
ImportData
InsertAfter
InsertBefore
InsertBreakAfter
InsertBreakBefore
InsertHyperlinkAfter
InsertHyperlinkBefore
InsertImageAfter
InsertImageBefore
InsertListAfter
InsertListBefore
InsertMergeFieldAfter
InsertMergeFieldBefore
InsertParagraphAfter
InsertParagraphBefore
InsertTableAfter
InsertTableBefore
InsertTextAfter
InsertTextBefore
Search
SearchAndReplace
SubElement

Field object

Properties
Code
Contents
Type

Methods
Update

Font object

Properties
Animation
Bold
Border
Caps
CharacterScaling
CharacterSpacing
DoubleStrike
Embossed
Engraved
FontName
FontSize
Hidden
Highlight
HyphenationRule
Italic
Kerning
NoProofChecking
Outlined
Shading
Shadow
SmallCaps
Strike
SuperSubScript
TextColor
Underline
UnderlineColor
VerticalPosition

HTMLToWord object

Properties
AcceptUnknownTags
DebugLogLevel
DefaultFont
HTMLInsertProperties
HTMLTagAction
IncludeContentsOfUnknownTags
InsertProperties
LogFileName
LoggingLevel

Methods
Format
FormatElement
GetImage
Image
Insert
InsertElement
InsertHTML

Hyperlink object

Methods
GetUrlString

InlineImage object

Properties
Height
Image
ImageFormat
Width

Methods
GetImage

List object

Properties
NumEntries

Methods
AddEntry
GetEntry
GetLevel
InsertEntry

ListEntry object

Properties
BulletText
LevelNum

Methods
RestartNumbering

ListLevel object

Properties
BulletText
Indent
NumberFormat
NumberIndent

MergeField object

Methods
GetFieldName

NamedStyle object

Properties
BuiltInStyleName
Font
Name
ParagraphFormatting
StyleType
TableFormatting

Paragraph object

Properties
Formatting
Style

ParagraphFormatting object

Properties
AbsolutePositioning
AfterAutoSpacing
AllowAutoHyphenation
BeforeAutoSpacing
KeepLinesTogether
KeepWithNext
LineNumberingAllowed
LineSpacing
LineSpacingRule
PageBreakBefore
Shading
SpaceAfter
SpaceBefore
TextJustification
WidowControl

Methods
GetBorder
GetIndent
SetIndent
SetLineSpacing

Position object

Properties
Position

Methods
InsertHyperlink
InsertImage
InsertList
InsertMergeField
InsertParagraph
InsertSection
InsertTable
InsertText

SearchMatch object

Properties
Element
GroupCount

Methods
Group
Replace

Section object

Properties
BorderAlwaysInFront
BorderAppliedTo
BorderOffsetsFromPage
BottomMargin
Break
FirstLineNum
FirstPageNum
FooterY
GutterWidth
HasTitlePage
HeaderY
LeftMargin
LineNumberingType
LineNumberX
LineNumModulus
LinesBetween
NumColumns
PageHeight
PageNumberFormat
PageOrientation
PageSize
PageWidth
RestartPageNum
RightMargin
TopMargin
Unlocked

Methods
GetBorder
GetFooter
GetHeader

Shading object

Properties

BackgroundColor
ForegroundColor
Pattern

Styles object

Properties
Item

Methods
Length

Table object

Properties
AbsolutePositioning
Formatting
Item
NumRows
PreferredWidth
Style

Methods
AddColumns
AddRows
CreateBookmarkOnRow
DeleteColumns
DeleteRows
GetNumColumns
GetPreferredColumnWidth
GetPreferredColumnWidthUnits
GetRowHeight
GetRowHeightExact
ImportDataRow
InsertColumns
InsertRows
MergeCells
GetPreferredColumnWidth
GetPreferredColumnWidthUnits
SetRowHeight
SetRowHeightExact

TableCell object

Properties
CellWidthUnits
FirstMerged
FirstVerticalMerged
Merged
NoWrap
PreferredWidth
Shading
VertAlignment
VerticalMerged

Methods
ApplyFont
ApplyParagraphFormatting
GetBorder
GetPadding
SetPadding

TableFormatting object

Properties
AutoWidth
CantSplit
DefaultShading
DefaultSpacing

Justification
LeftIndent
RepeatAsHeader

Method
GetDefaultBorder
GetDefaultPadding
SetDefaultPadding

TwipsConverter object

Methods
FromCentimeters
FromInches
FromPoints
ToCentimeters
ToInches
ToPoints

Watermark object

Properties
Color
FontName
Image
Scale
Semitransparent
Size
Text
Washout
WatermarkLayout
WatermarkType

Methods
ClearWatermark

WordApplication object

Properties
AlwaysUseUnicode
LicenseKey
Preserve
Version

Methods
Create
* Open
* Save

WordChangeTracking object

Properties
DisplayRevisions
LockRevisionMarkingState
PrintRevisions
ShowAnnotationRevisions
ShowFormattingRevisions
ShowInkRevisions
ShowInsertDeleteRevisions
ShowMarkupRevisions
TrackChanges

WordHyphenSettings object

Properties
AllowedConsecutiveHyphens
AutoHyphen
CapitalHyphenation

HyphenationWidth

WordViewSettings object

Properties
DisplayedOutlineLevels
DisplayFieldResults
DisplayHidden
DocumentView
EnvelopeVisible
IncludeFooterInBorder
IncludeHeaderInBorder
ShadeFormFields
Zoom
ZoomPercentage

WordWriterException object

WordTemplate Programmer's Reference

- WordTemplate — The WordTemplate object represents a WordWriter template. A WordWriter template is a file created in Microsoft Word that contains merge fields.

WordTemplate

Description

The WordTemplate object represents a WordWriter template. A WordWriter template is a file created in Microsoft Word that contains merge fields.

Signature: C#

```
public sealed class WordTemplate
```

Signature: VB.NET

```
Public NotInheritable Class WordTemplate
```

Remarks

The `WordTemplate` object is in the SoftArtisans.OfficeWriter.WordWriter namespace. The object can be referenced as `SoftArtisans.WordWriter.WordTemplate`. To minimize typing and errors, import the namespace to the aspx page, and reference the object as `WordTemplate`, without the namespace prefix.

Examples

If you are coding directly in the .aspx page, following the Page directive, include:

```
<%@ Import Namespace="SoftArtisans.OfficeWriter.WordWriter" %>
```

If you are coding in the "code behind" page (.aspx.vb or .aspx.cs), include an `Imports` or `using` statement at the top of the "code behind" page:

C#

```
using SoftArtisans.OfficeWriter.WordWriter;
```

vb.net

```
Imports SoftArtisans.OfficeWriter.WordWriter
```

To create an instance of the WordTemplate object use:

C#

```
WordTemplate wt = new WordTemplate();
```

vb.net

```
Dim wt As New WordTemplate()
```

Fields

ALL_ROWS	Used by SetDataSource to indicate that as much of the data source should be imported as possible.

Properties

Name	Description
Bookmarks	Returns the names of all bookmarks in a WordWriter template.
ContentType	Sets or returns the content-type header that will be sent to the browser with the generated Word document.
CultureInfo	The CultureInfo property allows you to override the server's default locale when generating a new Word file. For example, to generate a US English document on a French operating system, you could use `CultureInfo` to set the Word file's `System.Globalization.CultureInfo` to US English.
DataSourceSeparator	Use DataSourceSeparator to set the character that separates a data source name from a column name in a data marker. '.' is the default data source separator.
EnableNEXTFields	Gets or sets whether NEXT fields should be enabled in the WordTemplate. If NEXT fields are enabled, the data source will move to the next row of data when merge field is repeated in the template file.
FieldMarkers	Returns the names of all merge fields in the main document.
LicenseKey	Returns the registered 23-character license key that enables WordWriter.
PreserveFields	Gets or sets whether merge fields and next fields should be preserved after processing the WordTemplate.
SaveExtension	The extension that should be used when saving this template; based on the input file.
UpdateIfFields	Gets or sets whether IF fields should be evaluated server-side in the WordTemplate.
Version	Returns the exact version of WordWriter.

Methods

Name	Description
BookmarkFieldMarkers(String)	Returns the names of all merge fields in the specified bookmark.
Open(String)	Opens the WordWriter template specified by the `templateFileName` parameter.
Open(System.IO.Stream)	Opens a WordWriter template from a `System.IO.Stream`. A WordWriter template is a file created in Microsoft Word that contains merge fields.
Open(WordApplication, Document)	Passes a document from WordApplication to WordTemplate.
Process()	The Process method enters data source values in a template's merge fields, and creates an image of the output file (the new document) in memory.
Save(String)	Saves the generated Word file on the server.
Save(System.IO.Stream)	Sends the generated Word binary file to the specified `System.IO.Stream` or, a class derived from `System.IO.Stream` (for example, `System.IO.FileStream`).

Save(System.Web.HttpResponse)	If you pass Save an `HttpResponse` object object, WordWriter will stream the generated file to the client. If the user chooses to open (rather than save) the file, it will open in the browser window.
Save(System.Web.HttpResponse, String, Boolean)	If you pass `save` an HttpServletResponse object, WordWriter will stream the generated file to the client. This method allows you to specify a default client-side file name, and whether the file should be opened in the browser window or in Microsoft Word.
SetDataSource(Object(), String(), String)	Sets a main document data source to a one-dimensional object array. The method imports a single row of data to the specified set of main document merge fields.
SetDataSource(Object(), String())	Sets a main document data source to a one-dimensional object array. The method imports a single row of data to the specified set of main document merge fields.
SetDataSource(System.Data.DataSet, String)	Sets a main document data source to an ADO.NET DataSet. A DataSet contains a collection of DataTables. If you set a main document data source to a DataSet, WordWriter will get the first row of the first DataTable in the DataSet.
SetDataSource(System.Data.DataSet)	Sets a template's data source to an ADO.NET DataSet. A DataSet contains a collection of DataTables. If you set a template's data source to a DataSet, WordWriter will get the first row of the first DataTable in the DataSet.
SetDataSource(System.Data.DataTable, String)	Sets a main document data source to an ADO.NET DataTable. If the specified DataTable contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(System.Data.DataTable)	Sets a template's data source to an ADO.NET DataTable. If the specified DataTable contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(System.Data.IDataReader, String)	Sets a main document data source to an an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader, or an AdomdDataReader. If the specified DataReader returns more than one row, WordWriter will use the first row as the data source.
SetDataSource(System.Data.IDataReader)	Sets a template's data source to an an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader or an AdomdDataReader. If the specified DataReader returns more than one row, WordWriter will use the first row as the data source.
SetDataSource(Object, String)	Sets the data source for the specified main document merge field to an object. The **main document** is the part of the document that is not within a repeat block.
SetDataSource(System.Collections.IEnumerator, String())	Sets an `IEnumerator` collection as a data source to bind to a specified set of main document merge fields.
SetDataSource(System.Collections.IEnumerator, String(), String)	Sets an `IEnumerator` collection as a data source to bind to a specified set of main document merge fields.
SetDataSource(System.Collections.IDictionary)	Sets a template's main document data source to an `IDictionary` collection. An `IDictionary` represents a collection of key-and-value pairs.
SetDataSource(System.Collections.IDictionary, String)	Sets a template's main document data source to an `IDictionary` collection. An `IDictionary` represents a collection of key-and-value pairs.
SetDataSource(System.Collections.ICollection, String())	Sets an `ICollection` as a data source to bind to a specified set of main document merge fields.
SetDataSource(System.Collections.ICollection, String(), String)	Sets an `ICollection` collection as a data source to bind to a specified set of main document merge fields.
SetMailMerge(Object(,), String())	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(Object(,), String(), Int32, Boolean)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

SetMailMerge(Object(), String())	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(Object()(), String())	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(Object()(), String(), Int32, Boolean)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(System.Collections.ICollection)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the entire page as a repeat block, adding a new page when needed. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(System.Collections.IEnumerator)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(System.Data.DataSet)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.DataSet, Int32)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.DataTable)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.DataTable, Int32)	Sets the MailMerge's data source to a DataTable. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.IDataReader)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.IDataReader, Int32)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

SetRepeatBlock(System.Data.DataSet, String)	Sets the specified repeat block's data source to an ADO.NET DataSet. A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.
SetRepeatBlock(System.Data.DataSet, String, Int32)	Sets the specified repeat block's data source to an ADO.NET DataSet.
SetRepeatBlock(System.Data.DataTable, String)	Sets the specified repeat block's data source to an ADO.NET DataTable.
SetRepeatBlock(System.Data.DataTable, String, Int32)	Sets the specified repeat block's data source to an ADO.NET DataTable.
SetRepeatBlock(System.Data.IDataReader, String)	Sets the specified repeat block's data source to an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader or an AdomdDataReader.
SetRepeatBlock(System.Data.IDataReader, String, Int32)	Sets the specified repeat block's data source to an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader, or an AdomdDataReader.
SetRepeatBlock(Object(), String(), String)	Sets the specified repeat block's data source to a 1-dimensional array of objects.
SetRepeatBlock(Object(,), String(), String)	Sets the specified repeat block's data source to a multidimensional (rectangular) array of objects.
SetRepeatBlock(Object(,), String(), String, Int32, Boolean)	Sets the specified repeat block's data source to a multidimensional (rectangular) array of objects.
SetRepeatBlock(Object()(), String(), String)	Sets the specified repeat block's data source to a jagged array (array-of-arrays) of objects.
SetRepeatBlock(Object()(), String(), String, Int32, Boolean)	Sets the specified repeat block's data source to a jagged array (array-of-arrays) of objects.
SetRepeatBlock(System.Collections.IEnumerator, String)	Sets the specified repeat block's data source to an `IEnumerator` collection.
SetRepeatBlock(System.Collections.IEnumerator, String, String)	Sets the specified repeat block's data source to an `IEnumerator` collection.
SetRepeatBlock(System.Collections.ICollection, String)	Sets the specified repeat block's data source to an `ICollection`.
SetRepeatBlock(System.Collections.ICollection, String, String)	Sets the specified repeat block's data source to an `ICollection`.
WordTemplate()	Creates a new WordTemplate object.

Extension Methods

Introduced in build 4.5

Overload	Description
Open(Microsoft.SharePoint.SPDocumentLibrary, String)	Opens the WordWriter template specified by the `templateFileName` parameter.
Open(Microsoft.SharePoint.SPListItem, String)	Opens the WordWriter template specified by the `templateFileName` parameter.
Save(Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Saves the generated Word file to a SharePoint Document Library.
Save(Microsoft.SharePoint.SPListItem, String)	Saves the generated Word file to a SharePoint List Item.
SetDataSource(Microsoft.SharePoint.SPList, String)	Sets a main document data source to a SharePoint List. If the specified List contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(Microsoft.SharePoint.SPList)	Sets a main document data source to a SharePoint List. If the specified List contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)	Sets a main document data source to a SharePoint View. If the specified View contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Sets a main document data source to a SharePoint View. If the specified View contains more than one row, WordWriter will use the first row as the data source.

SetMailMerge(Microsoft.SharePoint.SPList, Int32)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(Microsoft.SharePoint.SPList)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, Int32)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetRepeatBlock(Microsoft.SharePoint.SPList, String, Int32)	Sets the specified repeat block's data source to a SharePoint List.
SetRepeatBlock(Microsoft.SharePoint.SPList, String)	Sets the specified repeat block's data source to a SharePoint List.
SetRepeatBlock(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, Int32)	Sets the specified repeat block's data source to a SharePoint View.
SetRepeatBlock(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)	Sets the specified repeat block's data source to a SharePoint View.

WordTemplate.ALL_ROWS

Description

Used by SetDataSource to indicate that as much of the data source should be imported as possible.

Signature: C#

```
public static const int ALL_ROWS = -1;
```

Signature: VB.NET

```
Public Shared Const ALLROWS As Integer = -1
```

WordTemplate.BookmarkFieldMarkers(String)

Description

Returns the names of all merge fields in the specified bookmark.

Signature: C#

```
public System.String[] BookmarkFieldMarkers(System.String aBookmark)
```

Signature: VB.NET

```
Public Function BookmarkFieldMarkers(ByVal aBookmark As String) As String()
```

Parameters

aBookmark

Returns

The names of all merge fields in the specified bookmark.

Remarks

Bookmarks are used to define repeat blocks. A repeat block is a fragment in the template document that contains merge fields and that will be repeated for each row in a data source.

Examples

C#

```
//--- Create the WordTemplate object.
          WordTemplate oWW = new WordTemplate();

          //--- Open a template using a string.
          oWW.Open(@"C:\template.doc");

          //--- Get all the bookmark names in the document.
          String[] allBookmarks = oWW.Bookmarks;

          //--- Get all the merge fields in the first bookmark.
          String[] someFields = oWW.BookmarkFieldMarkers(allBookmarks[0]);
```

vb.net

```
'--- Create the WordTemplate object.
          Dim oWW As New WordTemplate()

          '--- Open a template using a string.
          oWW.Open("C:\template.doc")

          '--- Get all the bookmark names in the document.
          Dim allBookmarks As String() = oWW.Bookmarks

          '--- Get all the merge fields in the first bookmark.
          Dim someFields As String() = oWW.BookmarkFieldMarkers(allBookmarks(0))
```

WordTemplate.Bookmarks

Description

Returns the names of all bookmarks in a WordWriter template.

Signature: C#

```
public System.String[] Bookmarks{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Bookmarks() As String()
```

Remarks

Bookmarks are used to define repeat blocks. A repeat block is a fragment in the template document that contains merge fields and that will be repeated for each row in a data source.

Examples

C#

```
//--- Create the WordTemplate object.
          WordTemplate oWW = new WordTemplate();

          //--- Open a template using a string.
          oWW.Open("C:\template.doc");

          //--- Get all the bookmark names in the document.
          String[] allBookmarks = oWW.Bookmarks;
```

vb.net

```
'--- Create the WordTemplate object.
          Dim oWW As New WordTemplate()

          '--- Open a template using a string.
          oWW.Open("C:\template.doc")

          '--- Get all the bookmark names in the document.
          Dim allBookmarks As String() = oWW.Bookmarks
```

WordTemplate.ContentType

Description

Sets or returns the content-type header that will be sent to the browser with the generated Word document.

Signature: C#

```
public System.String ContentType{ get; set; }
```

Signature: VB.NET

```
Public Property ContentType() As String
```

Remarks

ContentType must not be used before Open is called, as the content-type of the file cannot be known until after the file is opened. Getting or setting ContentType before calling Open will throw an exception.

Examples

C#

```
...
          WordTemplate WordTempl = new WordTemplate();
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"));
          WordTempl.SetDataSource(MyDataTable);

          //--- Set the ContentType
          //--- ContentType is application/vns.ms-word by default for
          //--- .doc files, while the default ContentType for
          //--- .docx files is
          //--- application/vnd.openxmlformats-officedocument.wordprocessingml.document
          //--- and for .docm files is
          //--- application/vnd.ms-word.document.macroEnabled.12
          WordTempl.ContentType = "application/unknown";
          WordTempl.Process();
          WordTempl.Save(Page.Response, "BasicGenerated.doc", false);
          ...
```

vb.net

```
...
          Dim WordTempl As New WordTemplate()
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"))
          WordTempl.SetDataSource(MyDataTable)

          '--- Set the ContentType
          '--- ContentType is application/vns.ms-word by default for
          '--- .doc files, while the default ContentType for
          '--- .docx files is
          '--- application/vnd.openxmlformats-officedocument.wordprocessingml.document
          '--- and for .docm files is
          '--- application/vnd.ms-word.document.macroEnabled.12
          WordTempl.ContentType = "application/unknown"
          WordTempl.Process()
          WordTempl.Save(Page.Response, "BasicGenerated.doc", False)
          ...
```

WordTemplate.CultureInfo

Description

The CultureInfo property allows you to override the server's default locale when generating a new Word file. For example, to generate a US English document on a French operating system, you could use `CultureInfo` to set the Word file's `System.Globalization.CultureInfo` to US English.

Signature: C#

```
public System.Globalization.CultureInfo CultureInfo{ set; }
```

Signature: VB.NET

```
Public WriteOnly Property CultureInfo() As System.Globalization.CultureInfo
```

Remarks

Set `CultureInfo` to a `System.Globalization.CultureInfo` object. This object represents information about a culture including its alphabet, calendar, and date formats. When setting `CultureInfo`, import the `System.Globalization` namespace to the ASP.NET page, for example, to import the namespace to a code-behind page, use:

C#

```
using System.Globalization;
```

vb.net

```
Imports System.Globalization
```

`CultureInfo` must include both language and country (for example "fr-BE" for French-Belgium).

See `System.Globalization.CultureInfo` for more information about the class, its members, and a list of CultureInfo codes.

Examples

C#

```
using System.Globalization;
          ...
          string[] NamesArr = {"Name", "Company", "DateTime"};
          object[] ValuesArr = {Name, Company, DateTime.Now};
          ...
          WordTemplate WordTempl = new WordTemplate();

          //--- Create a new CultureInfo object and
          //--- set WordTemplate's CultureInfo property
          CultureInfo enUS = new CultureInfo("en-US");
          WordTempl.CultureInfo = enUS;
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"));
          WordTempl.SetDataSource(ValuesArr, NamesArr);
          WordTempl.Process();
          WordTempl.Save(Page.Response, "BasicGenerated.doc", false);
          ...
```

vb.net

```
Imports System.Globalization
          ...
          Dim NamesArr As String() = {"Name", "Company", "DateTime"}
          Dim ValuesArr As Object() = {Name, Company, DateTime.Now}
          ...
          Dim WordTempl As New WordTemplate()

          '--- Create a new CultureInfo object and
          '--- set WordTemplate's CultureInfo property
          Dim enUS As New CultureInfo("en-US")
          WordTempl.CultureInfo = enUS
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"))
          WordTempl.SetDataSource(ValuesArr, NamesArr)
          WordTempl.Process()
          WordTempl.Save(Page.Response, "BasicGenerated.doc", False)
          ...
```

WordTemplate.DataSourceSeparator

Description

Use DataSourceSeparator to set the character that separates a data source name from a column name in a data marker. '.' is the default data source separator.

Signature: C#

```
public char DataSourceSeparator{ get; set; }
```

Signature: VB.NET

```
Public Property DataSourceSeparator() As Char
```

Remarks

The `DataSourceSeparator` property must be set **before** Open is called because the merge field collection is parsed when the document is opened.

The following characters cannot be used as data source separators:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890_@$#()[],
```

The space character cannot be used as a data source separator.

Examples

C#

```
WordTemplate oWW = new WordTemplate();
          oWW.DataSourceSeparator = "-";
          oWW.Open("Template.doc");
          ...
```

vb.net

```
Dim oWW As New WordTemplate()
          oWW.DataSourceSeparator = "-"
          oWW.Open("Template.doc")
          ...
```

WordTemplate.EnableNEXTFields

Description

Gets or sets whether NEXT fields should be enabled in the WordTemplate. If NEXT fields are enabled, the data source will move to the next row of data when merge field is repeated in the template file.

Signature: C#

```
public boolean EnableNEXTFields{ get; set; }
```

Signature: VB.NET

```
Public Property EnableNEXTFields() As Boolean
```

Examples

C#

```
WordTemplate WordTempl = new WordTemplate();
          WordTempl.EnableNEXTFields = true;
```

vb.net

```
Dim WordTempl As New WordTemplate()
          WordTempl.EnableNEXTFields = True
```

WordTemplate.FieldMarkers

Description

Returns the names of all merge fields in the main document.

Signature: C#

```
public System.String[] FieldMarkers{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property FieldMarkers() As String()
```

Remarks

The main document is any part of the template that is not within a repeat block. (A repeat block is a fragment in the template document - defined by a Word bookmark - that contains merge fields and that will be repeated for each row in a data source.)

Examples

C#

```
//--- Create the WordTemplate object.
          WordTemplate oWW = new WordTemplate();

          //--- Open a template using a string.
          oWW.Open(@"C:\template.doc");

          //--- Get all the merge fields in the document.
          String[] allMergefields = oWW.FieldMarkers;
```

vb.net

```
'--- Create the WordTemplate object.
          Dim oWW As New WordTemplate()

          '--- Open a template using a string.
          oWW.Open("C:\template.doc")

          '--- Get all the merge fields in the document.
          Dim allMergefields As String() = oWW.FieldMarkers
```

WordTemplate.LicenseKey

Description

Returns the registered 23-character license key that enables WordWriter.

Signature: C#

```
public System.String LicenseKey{ get; set; }
```

Signature: VB.NET

```
Public Property LicenseKey() As String
```

Remarks

The key contains information about WordWriter, including product name, platform, and version. Only the 23-character key is stored in the registry. To determine which licensed version is currently installed on the server, run LicenseManager.exe (in OfficeWriter\WordWriter) and enter the license key retrieved by the LicenseKey property.

Examples

C#

```
WordTemplate WordTempl = new WordTemplate();
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"));
          string key = WordTempl.LicenseKey;
          string[] NamesArr = {"LicenseKey", "DateTime"};
          object[] ValuesArr = {key, DateTime.Now};
          WordTempl.SetDataSource(ValuesArr, NamesArr);
          WordTempl.Process();
          WordTempl.Save(Page.Response, "BasicGenerated.doc", false);
```

vb.net

```
Dim WordTempl As New WordTemplate()
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"))
          Dim key As String = WordTempl.LicenseKey
          Dim NamesArr As String() = {"LicenseKey", "DateTime"}
          Dim ValuesArr As Object() = {key, DateTime.Now}
          WordTempl.SetDataSource(ValuesArr, NamesArr)
          WordTempl.Process()
          WordTempl.Save(Page.Response, "BasicGenerated.doc", False)
```

WordTemplate.Open

Overload	Description
Open(String)	Opens the WordWriter template specified by the `templateFileName` parameter.
Open(System.IO.Stream)	Opens a WordWriter template from a `System.IO.Stream`. A WordWriter template is a file created in Microsoft Word that contains merge fields.
Open(WordApplication, Document)	Passes a document from WordApplication to WordTemplate.

SharePoint Integration Extension Methods

Introduced in build 4.5

Overload	Description
Open(Microsoft.SharePoint.SPDocumentLibrary, String)	Opens the WordWriter template specified by the `templateFileName` parameter.
Open(Microsoft.SharePoint.SPListItem, String)	Opens the WordWriter template specified by the `templateFileName` parameter.

WordTemplate.Open(String)

Description

Opens the WordWriter template specified by the `templateFileName` parameter.

Signature: C#

```
public void Open(System.String templateFileName)
```

Signature: VB.NET

```
Public Sub Open(ByVal templateFileName As String)
```

Parameters

templateFileName

The complete path and file name of the template Word file.

Exceptions

ArgumentNullException FileNotFoundException

Remarks

A WordWriter template is a file created in Microsoft Word that contains merge fields. WordWriter supports Microsoft Word 97, 2000, 2002 (XP), 2003, and 2007. Do not use Open to open files created in earlier versions of Microsoft Word.

> WordWriter templates can be created in Windows 2003, however, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.

Examples

C#

```
oWW.Open(@"c:\templates\StringTemplate.doc");
```

vb.net

```
oWW.Open("c:\templates\StringTemplate.doc")
```

WordTemplate.Open(System.IO.Stream)

Description

Opens a WordWriter template from a `System.IO.Stream`. A WordWriter template is a file created in Microsoft Word that contains merge fields.

Signature: C#

```
public void Open(System.IO.Stream templateStream)
```

Signature: VB.NET

```
Public Sub Open(ByVal templateStream As System.IO.Stream)
```

Parameters

templateStream

A `System.IO.Stream` object, or a class derived from System.IO.Stream (for example, System.IO.FileStream).

Exceptions

ArgumentNullException

Remarks

WordWriter supports Microsoft Word 97, 2000, 2002 (XP), 2003, and 2007. Do not use Open to open files created in earlier versions of Microsoft Word.

> WordWriter templates can be created in Windows 2003, however, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.

Examples

C#

```
//--- FileStream and FileMode are in System.IO
          FileStream oFileStream = new FileStream(@"c:\templates\StringTemplate.doc",
               FileMode.Open);
          oWW.Open(oFileStream);
```

vb.net

```
'--- FileStream and FileMode are in System.IO
          Dim oFileStream As New FileStream("c:\templates\StringTemplate.doc", _
               FileMode.Open, FileAccess.Read)
          oWW.Open(oFileStream)
```

WordTemplate.Open(WordApplication, Document)

Description

Passes a document from WordApplication to WordTemplate.

Signature: C#

```
public void Open(WordApplication aApp, Document aDoc)
```

Signature: VB.NET

```
Public Sub Open(ByVal aApp As WordApplication, ByVal aDoc As Document)
```

Parameters

aApp

The instance of `WordApplication` that was used to create the document.

aDoc

A Document object representing the Word file created with WordApplication.

Remarks

You can use `WordApplication` to create a document with merge fields (a template) and pass the file to `WordTemplate` to populate the merge fields.

Examples

C#

```
WordApplication wwApp = new WordApplication();
          Document doc = wwApp.Create();
          WordTemplate wwTmpl = New WordTemplate();
          wwTmpl.Open(wwApp, doc);
```

vb.net

```
Dim wwApp As New WordApplication()
          Dim doc As Document = wwApp.Create()
          Dim wwTmpl As New WordTemplate()
          wwTmpl.Open(wwApp, doc)
```

WordTemplate.PreserveFields

Description

Gets or sets whether merge fields and next fields should be preserved after processing the WordTemplate.

Signature: C#

```
public boolean PreserveFields{ get; set; }
```

Signature: VB.NET

```
Public Property PreserveFields() As Boolean
```

Remarks

If set to `true`, all merge fields and next fields on the template will be preserved by the WordTemplate object after processing and will be available for future use. If set to `false`, the merge fields and next fields will be replaced with data during processing and will not be available for future use. PreserveFields must be set before calling Open

Examples

C#

```
WordTemplate WordTempl = new WordTemplate();
          WordTempl.PreserveFields = true;
```

vb.net

```
Dim WordTempl As New WordTemplate()
          WordTempl.PreserveFields = True
```

WordTemplate.Process()

Description

The Process method enters data source values in a template's merge fields, and creates an image of the output file (the new document) in memory.

Signature: C#

```
public void Process()
```

Signature: VB.NET

```
Public Sub Process()
```

Remarks

The save method can then save the output file to disk, stream it to the browser, or both.

Call `Process` after SetDataSource and before Save.

You can only call `Process` once for each instance of WordTemplate.

Examples

Generate a single Word file:

C#

```
//--- Create an object array of values
            //--- and a string array of field names to pass to
            //--- SetDataSource.
            Object[] arrValue = {"SoftArtisans WordWriter"};
            string[] arrName = {"ProductName"};

            //--- Create an instance of WordTemplate.
            WordTemplate oWW = new WordTemplate();

            //--- Open a template file.
            oWW.Open(Server.MapPath(@"../StringVarTemplate.doc"));

            //--- Set the file's data source by passing
            //--- SetDataSource an array of values and an
            //--- array of merge field names.
            oWW.SetDataSource(arrValue, arrName);

            //--- Call Process() to populate the template with
            //--- the new data.
            oWW.Process();

            //--- After processing the template, call Save() and pass it
            //--- Page.Response to stream the generated file to the browser.
            oWW.Save(Page.Response, "StringVarOutput.doc", false);
```

vb.net

```
'--- Create an object array of values
            '--- and a string array of field names to pass to
            '--- SetDataSource.
            Dim arrValue As Object() = {"SoftArtisans WordWriter"}
            Dim arrName As String() = {"ProductName"}

            '--- Create an instance of WordTemplate.
            Dim oWW As New WordTemplate()

            '--- Open a template file.
            oWW.Open(Server.MapPath("./StringVarTemplate.doc"))

            '--- Set the file's data source by passing
            '--- SetDataSource an array of values and an
            '--- array of merge field names.
            oWW.SetDataSource(arrValue, arrName)

            '--- Call Process() to populate the template with
            '--- the new data.
            oWW.Process()

            '--- After processing the template, call Save() and pass it
            '--- Page.Response to stream the generated file to the browser.
            oWW.Save(Page.Response, "StringVarOutput.doc", False)
```

WordTemplate.Save

Overload	Description
Save(String)	Saves the generated Word file on the server.
Save(System.IO.Stream)	Sends the generated Word binary file to the specified `System.IO.Stream` or, a class derived from `System.IO.Stream` (for example, `System.IO.FileStream`).
Save(System.Web.HttpResponse)	If you pass Save an `HttpResponse` object object, WordWriter will stream the generated file to the client. If the user chooses to open (rather than save) the file, it will open in the browser window.

Save(System.Web.HttpResponse, String, Boolean)	If you pass `save` an HttpServletResponse object, WordWriter will stream the generated file to the client. This method allows you to specify a default client-side file name, and whether the file should be opened in the browser window or in Microsoft Word.

SharePoint Integration Extension Methods

Introduced in build 4.5

Overload	Description
Save(Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Saves the generated Word file to a SharePoint Document Library.
Save(Microsoft.SharePoint.SPListItem, String)	Saves the generated Word file to a SharePoint List Item.

WordTemplate.Save(String)

Description

Saves the generated Word file on the server.

Signature: C#

```
public void Save(System.String outputFileName)
```

Signature: VB.NET

```
Public Sub Save(ByVal outputFileName As String)
```

Parameters

strFilePath

Specifies a complete path and file name for the generated file. WordWriter will save the file to this location. If a file with the same name exists, it will be overwritten by the new Word file.

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

Remarks

You can call Save more than once for a single instance of WordTemplate. This allows you to save more than one copy of a generated file and/or save the file on the server and stream it to the client.

Examples

C#

```
WordTemplate oWW = new WordTemplate();
          oWW.Open(@"c:\templates\Template.doc");
          oWW.SetDataSource(arrValue, arrName);
          oWW.Process();
          oWW.Save(@"c:\reports\Report.doc");
```

vb.net

```
Dim oWW As New WordTemplate()
          oWW.Open("c:\templates\Template.doc")
          oWW.SetDataSource(arrValue, arrName)
          oWW.Process()
          oWW.Save("c:\reports\Report.doc")
```

WordTemplate.Save(System.IO.Stream)

Description

Sends the generated Word binary file to the specified `System.IO.Stream` or, a class derived from `System.IO.Stream` (for example, `System.IO.FileStream`).

Signature: C#

```
public void Save(System.IO.Stream outputStream)
```

Signature: VB.NET

```
Public Sub Save(ByVal outputStream As System.IO.Stream)
```

Parameters

oStreamObj

A `System.IO.Stream` object (for example, `Response.OutputStream`) or, a class derived from `System.IO.Stream` (for example, `System.IO.FileStream`)

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

Remarks

If you pass Save a `System.IO.FileStream`, WordWriter will save the generated file on the server. If you pass `Save` `Response.OutputStream`, WordWriter will stream the the generated file to the client.

You can call `Save` more than once for a single instance of WordTemplate. This allows you to save more than one copy of a generated file, and/or both save the file on the server and stream it to the client.

Examples

C#

```
//--- Open a FileStream object, pass it to
          //--- the Save method, and close the FileStream object.
          FileStream oFileStream = new FileStream(@"c:\temp\StreamOutput.doc", FileMode.Create);
          oWW.Save(oFileStream);

          //--- Could put this in a finally block
          if(oFileStream!=null)
               oFileStream.Close();
```

vb.net

```
'--- Open a FileStream  object, pass it to
          '--- the Save method, and close the FileStream object.
          Dim oFileStream As New FileStream("c:\temp\StreamOutput.doc", _
               FileMode.Create)
          oWW.Save(oFileStream)

          '--- Could put this in a finally block
          If Not oFileStream Is Nothing Then
               oFileStream.Close()
          End If
```

WordTemplate.Save(System.Web.HttpResponse, String, Boolean)

Description

If you pass `save` an HttpServletResponse object, WordWriter will stream the generated file to the client. This method allows you to specify a default client-side file name, and whether the file should be opened in the browser window or in Microsoft Word.

Signature: C#

```
public void Save(System.Web.HttpResponse response, System.String attachmentName, boolean
openInBrowser)
```

Signature: VB.NET

```
Public Sub Save(ByVal response As System.Web.HttpResponse, ByVal attachmentName As String, ByVal
openInBrowser As Boolean)
```

Parameters

response

The `HttpResponse` instance of the page.

attachmentName

Specifies a name for the generated Word file; this name will be displayed in the download dialog when the file is streamed to the browser.

openInBrowser

If `openInBrowser` is set to `true`, and the user chooses to open the file, the file will open in the browser window. If `openInBrowser` is set to `false`, and the user chooses to open the file, the file will open in Microsoft Word. By default, the file will open in the browser window.

> ⚠ Not all browsers can embed a Word file in the browser window.

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

Remarks

You can call Save more than once for a single instance of WordTemplate. This allows you to save more than one copy of a generated file, and/or both save the file on the server and stream it to the client.

Examples

Stream to the client and open in MS Word

When you pass an HttpResponse object to `Save`, WordWriter will stream the generated Word file to the client. The browser will display a File Download dialog asking the user to open or save the file. The method's second parameter specifies a file name to display in the File Download dialog. If the method's third parameter - OpenInBrowser - is False and the user chooses to open the Word file, the file will open in Microsoft Word.

C#

```
oWW.Save(Page.Response, "StringVarOutput.doc", false);
```

vb.net

```
oWW.Save(Page.Response, "StringVarOutput.doc", False)
```

Stream to the client and open in the browser window

When you pass an HttpResponse object to `Save`, WordWriter will stream the generated Word file to the client. The browser will display a File Download dialog asking the user to open or save the file. The method's second parameter specifies a file name to display in the File Download dialog. If the method's third parameter - `OpenInBrowser` - is True and the user chooses to open the Word file, the file will open in the browser window.

C#

```
oWW.Save(Page.Response, "StringVarOutput.doc", true);
```

vb.net

```
oWW.Save(Page.Response, "StringVarOutput.doc", True)
```

WordTemplate.Save(System.Web.HttpResponse)

Description

If you pass Save an `HttpResponse` object object, WordWriter will stream the generated file to the client. If the user chooses to open (rather than save) the file, it will open in the browser window.

Signature: C#

```
public void Save(System.Web.HttpResponse response)
```

Signature: VB.NET

```
Public Sub Save(ByVal response As System.Web.HttpResponse)
```

Parameters

response

The page's `HttpResponse` object.

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

Remarks

You can call `Save` more than once for a single instance of WordTemplate. This allows you to save more than one copy of a generated file, and/or both save the file on the server and stream it to the client.

Examples

C#

```
...
          WordTemplate oWW = new WordTemplate();
          oWW.Open(Server.MapPath(@"./StringVarTemplate.doc"));
          oWW.SetDataSource(MyDataTable);
          oWW.Process();
          oWW.Save(Page.Response);
```

vb.net

```
...
          Dim oWW As New WordTemplate()
          oWW.Open(Server.MapPath("./StringVarTemplate.doc"))
          oWW.SetDataSource(MyDataTable)
          oWW.Process()
          oWW.Save(Page.Response)
```

WordTemplate.SaveExtension

Introduced in build 4.1.0.1322

Description

The extension that should be used when saving this template; based on the input file.

Signature: C#

```
public String SaveExtension { get; }
```

Signature: VB.NET

```
Public ReadOnly Property SaveExtension() As [String]
```

Remarks

The extension does not include the dot, such as "docx" or "dotm". Until a file is opened this will be null.

Examples

C#

```
wordTemplate.Save("output." + wordTemplate.SaveExtension);
```

vb.net

```
wordTemplate.Save("output." & wordTemplate.SaveExtension);
```

WordTemplate.SetDataSource

Overload	Description
SetDataSource(Object(), String(), String)	Sets a main document data source to a one-dimensional object array. The method imports a single row of data to the specified set of main document merge fields.
SetDataSource(Object(), String())	Sets a main document data source to a one-dimensional object array. The method imports a single row of data to the specified set of main document merge fields.
SetDataSource(System.Data.DataSet, String)	Sets a main document data source to an ADO.NET DataSet. A DataSet contains a collection of DataTables. If you set a main document data source to a DataSet, WordWriter will get the first row of the first DataTable in the DataSet.
SetDataSource(System.Data.DataSet)	Sets a template's data source to an ADO.NET DataSet. A DataSet contains a collection of DataTables. If you set a template's data source to a DataSet, WordWriter will get the first row of the first DataTable in the DataSet.
SetDataSource(System.Data.DataTable, String)	Sets a main document data source to an ADO.NET DataTable. If the specified DataTable contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(System.Data.DataTable)	Sets a template's data source to an ADO.NET DataTable. If the specified DataTable contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(System.Data.IDataReader, String)	Sets a main document data source to an an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader, or an AdomdDataReader. If the specified DataReader returns more than one row, WordWriter will use the first row as the data source.
SetDataSource(System.Data.IDataReader)	Sets a template's data source to an an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader or an AdomdDataReader. If the specified DataReader returns more than one row, WordWriter will use the first row as the data source.
SetDataSource(Object, String)	Sets the data source for the specified main document merge field to an object. The **main document** is the part of the document that is not within a repeat block.
SetDataSource(System.Collections.IEnumerator, String())	Sets an `IEnumerator` collection as a data source to bind to a specified set of main document merge fields.

SetDataSource(System.Collections.IEnumerator, String(), String)	Sets an `IEnumerator` collection as a data source to bind to a specified set of main document merge fields.
SetDataSource(System.Collections.IDictionary)	Sets a template's main document data source to an `IDictionary` collection. An `IDictionary` represents a collection of key-and-value pairs.
SetDataSource(System.Collections.IDictionary, String)	Sets a template's main document data source to an `IDictionary` collection. An `IDictionary` represents a collection of key-and-value pairs.
SetDataSource(System.Collections.ICollection, String())	Sets an `ICollection` as a data source to bind to a specified set of main document merge fields.
SetDataSource(System.Collections.ICollection, String(), String)	Sets an `ICollection` collection as a data source to bind to a specified set of main document merge fields.

SharePoint Integration Extension Methods

Introduced in build 4.5

Overload	Description
SetDataSource(Microsoft.SharePoint.SPList, String)	Sets a main document data source to a SharePoint List. If the specified List contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(Microsoft.SharePoint.SPList)	Sets a main document data source to a SharePoint List. If the specified List contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)	Sets a main document data source to a SharePoint View. If the specified View contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Sets a main document data source to a SharePoint View. If the specified View contains more than one row, WordWriter will use the first row as the data source.

WordTemplate.SetDataSource(Object, String)

Description

Sets the data source for the specified main document merge field to an object. The **main document** is the part of the document that is not within a repeat block.

Signature: C#

```
public void SetDataSource(System.Object value, System.String dataSourceName)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal value As Object, ByVal dataSourceName As String)
```

Parameters

value

An object to use as a data source. WordWriter will insert this value in the specified merge field.

name

The name of the the merge field at which to insert the value.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

The data marker requires a #1 on the template. For example, to use the sample code below, you must have a merge field on the template with the

data marker "FirstName.#1".

Examples

C#

```
WordTemplate template = new WordTemplate();
          template.Open(Request.PhysicalApplicationPath + "file.doc");
          string name = txtName.Text;
          template.SetDataSource(name,"FirstName");
          template.Process();
          template.Save(Response,"file.doc",false);
          ...
```

vb.net

```
Dim template As New WordTemplate()
          template.Open(Request.PhysicalApplicationPath & "file.doc")
          Dim name As String = txtName.Text
          template.SetDataSource(name, "FirstName")
          template.Process()
          template.Save(Response, "file.doc", False)
```

WordTemplate.SetDataSource(Object(), String(), String)

Description

Sets a main document data source to a one-dimensional object array. The method imports a single row of data to the specified set of main document merge fields.

Signature: C#

```
public void SetDataSource(System.Object[] dataRow, System.String[] fieldNames, System.String name)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dataRow As Object(), ByVal fieldNames As String(), ByVal name As
String)
```

Parameters

data

An object array of values to use as a data source. WordWriter will insert these values in the template's merge fields.

fieldNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template

name

A string that matches the data source name.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

The method's third parameter - `name` - specifies the data source name for the set of merge fields. For example, if the main document contains the following sets of merge fields:

«Customer.FirstName»	«Order.OrderId»
«Customer.LastName»	«Order.Date»

For the first set of merge fields, set `name` to "Customer," and for the second, set `name` to "Order."

SetDataSource takes an object array of values and a string array of field names. The two arrays must contain the same number of elements. Each name in the array of field names must be the same as the corresponding merge field name in the template.

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in a set (merge fields that share the same data source name or number) may not exceed the number of values in the data source defined by `SetDataSource`. However, the number of values in the data source may be greater than the number of merge fields in the set.

This method may be called once for each set of merge fields in the main document.

Examples

C#

```
//--- Create a string array of field names.
          //--- The field names must be the same as the merge field
          //--- names in the template.
          string[] arrFields = {"CompanyName",
               "StreetAddr",
               "City",
               "State",
               "ZipCode"};

          //--- Create an object array of values
          //--- Ordinal numbers match those of the fields array
          object[] arrValues = {"SoftArtisans",
               "1330 Beacon Street",
               "Brookline",
               "MA",
               "02446"};
          oWW.SetDataSource(arrValues, arrFields, "data");
```

vb.net

```
'--- Create a string array of field names.
          '--- The field names  must be the same as the merge field
          '--- names in the template.
          Dim arrFields As String() = {"CompanyName", _
               "StreetAddr", _
               "City", _
               "State", _
               "ZipCode"}
          '--- Create an object array of values
          '--- Ordinal numbers match those of the fields array
          Dim arrValues As Object() = {"SoftArtisans", _
               "1330 Beacon Street", _
               "Brookline", _
               "MA", _
               "02446"}
          oWW.SetDataSource(arrValues, arrFields, "data")
```

WordTemplate.SetDataSource(Object(), String())

Description

Sets a main document data source to a one-dimensional object array. The method imports a single row of data to the specified set of main document merge fields.

Signature: C#

```
public void SetDataSource(System.Object[] dataRow, System.String[] fieldNames)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dataRow As Object(), ByVal fieldNames As String())
```

Parameters

data

An object array of values to use as a data source. WordWriter will insert these values in the template's merge fields.

fieldNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

SetDataSource takes an object array of values and a string array of field names. The two arrays must contain the same number of elements. Each name in the array of field names must be the same as the corresponding merge field name in the template.

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in a set (merge fields that share the same data source name or number) may not exceed the number of values in the data source defined by `SetDataSource`. However, the number of values in the data source may be greater than the number of merge fields in the set.

This method may be called once for each set of merge fields in the main document.

Examples

C#

```
//--- Create a string array of field names.
          //--- The field names must be the same as the merge field
          //--- names in the template.
          string[] arrFields = {"CompanyName",
               "StreetAddr",
               "City",
               "State",
               "ZipCode"};

          //--- Create an object array of values
          //--- Ordinal numbers match those of the fields array
          object[] arrValues = {"SoftArtisans",
               "1330 Beacon Street",
               "Brookline",
               "MA",
               "02446"};
          oWW.SetDataSource(arrValues, arrFields);
```

vb.net

```
'--- Create a string array of field names.
          '--- The field names  must be the same as the merge field
          '--- names in the template.
          Dim arrFields As String() = {"CompanyName", _
               "StreetAddr", _
               "City", _
               "State", _
               "ZipCode"}

          '--- Create an object array of values
          '--- Ordinal numbers match those of the fields array
          Dim arrValues As Object() = {"SoftArtisans", _
               "1330 Beacon Street", _
               "Brookline", _
               "MA", _
               "02446"}
          oWW.SetDataSource(arrValues, arrFields)
```

WordTemplate.SetDataSource(System.Collections.ICollection, String(), String)

Description

Sets an `ICollection` collection as a data source to bind to a specified set of main document merge fields.

Signature: C#

```
public void SetDataSource(System.Collections.ICollection dataSource, System.String[] fieldNames,
System.String dataSourceName)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dataSource As System.Collections.ICollection, ByVal fieldNames As
String(), ByVal dataSourceName As String)
```

Parameters

dataSource

An `ICollection` to use as the data source.

fieldNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template.

dataSourceName

Specifies the data source name for the set of merge fields.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

This method takes a collection of values and an array of field names. The two must contain the same number of elements. Each name in the array of field names must be the same as the corresponding merge field name in the template.

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the main template document (the part of the document that is not within a repeat block) may not exceed the number of values in the data source defined by SetDataSource. However, the number of values in the data source may be greater than the number of merge fields in the main template document.

Examples

C#

```
wordTempl.SetDataSource(collValues, arrNames, "Products");
```

vb.net

```
wordTempl.SetDataSource(collValues, arrNames, "Products")
```

WordTemplate.SetDataSource(System.Collections.ICollection, String())

Description

Sets an `ICollection` as a data source to bind to a specified set of main document merge fields.

Signature: C#

```
public void SetDataSource(System.Collections.ICollection dataSource, System.String[] fieldNames)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dataSource As System.Collections.ICollection, ByVal fieldNames As
String())
```

Parameters

dataSource

An `ICollection` to use as the data source.

fieldNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

This method takes a collection of values and an array of field names. The two must contain the same number of elements. Each name in the array of field names must be the same as the corresponding merge field name in the template.

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the main template document (the part of the document that is not within a repeat block) may not exceed the number of values in the data source defined by SetDataSource. However, the number of values in the data source may be greater than the number of merge fields in the main template document.

Examples

C#

```
wordTempl.SetDataSource(collValues, arrNames);
```

vb.net

```
wordTempl.SetDataSource(collValues, arrNames)
```

WordTemplate.SetDataSource(System.Collections.IDictionary, String)

Description

Sets a template's main document data source to an `IDictionary` collection. An `IDictionary` represents a collection of key-and-value pairs.

Signature: C#

```
public void SetDataSource(System.Collections.IDictionary dataSource, System.String dataSourceName)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dataSource As System.Collections.IDictionary, ByVal dataSourceName As
String)
```

Parameters

dataSource

An `IDictionary` collection of key-and-value pairs.

datSourceName

The name of the set of template merge fields at which to insert the values imported from the data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the main template document (the part of the document that is not within a repeat block) may not exceed the number of values in the data source defined by SetDataSource. However, the number of values in the data source may be greater than the number of merge fields in the main template document.

Examples

C#

```
wTempl.SetDataSource(coll, "Products");
```

vb.net

```
wTempl.SetDataSource(coll, "Products")
```

WordTemplate.SetDataSource(System.Collections.IDictionary)

Description

Sets a template's main document data source to an `IDictionary` collection. An `IDictionary` represents a collection of key-and-value pairs.

Signature: C#

```
public void SetDataSource(System.Collections.IDictionary dataSource)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dataSource As System.Collections.IDictionary)
```

Parameters

dataSource

An `IDictionary` collection of key-and-value pairs.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the main template document (the part of the document that is not within a repeat block) may not exceed the number of values in the data source defined by SetDataSource. However, the number of values in the data source may be greater than the number of merge fields in the main template document.

Do not call this method more than once for a single instance of WordTemplate. To set multiple main document data sources, use the following methods:
SetDataSource(Object(), String(), String)
SetDataSource(System.Data.DataSet, String)
SetDataSource(System.Data.DataTable, String)
SetDataSource(System.Data.IDataReader, String)

Examples

C#

```
wTempl.SetDataSource(coll);
```

vb.net

```
wTempl.SetDataSource(coll)
```

WordTemplate.SetDataSource(System.Collections.IEnumerator, String(), String)

Description

Sets an `IEnumerator` collection as a data source to bind to a specified set of main document merge fields.

Signature: C#

```
public void SetDataSource(System.Collections.IEnumerator dataSource, System.String[] fieldNames,
System.String dataSourceName)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dataSource As System.Collections.IEnumerator, ByVal fieldNames As
String(), ByVal dataSourceName As String)
```

Parameters

dataSource

An `IEnumerator` collection to use as the data source.

fieldNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template.

dataSourceName

Specifies the data source name for the set of merge fields.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

This method takes a collection of values and an array of field names. The two must contain the same number of elements. Each name in the array of field names must be the same as the corresponding merge field name in the template.

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the main template document (the part of the document that is not within a repeat block) may not exceed the number of values in the data source defined by SetDataSource. However, the number of values in the data source may be greater than the number of merge fields in the main template document.

Examples

C#

```
wordTempl.SetDataSource(collValues, arrNames, "Products");
```

vb.net

```
wordTempl.SetDataSource(collValues, arrNames, "Products")
```

WordTemplate.SetDataSource(System.Collections.IEnumerator, String())

Description

Sets an `IEnumerator` collection as a data source to bind to a specified set of main document merge fields.

Signature: C#

```
public void SetDataSource(System.Collections.IEnumerator dataSource, System.String[] fieldNames)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dataSource As System.Collections.IEnumerator, ByVal fieldNames As
String())
```

Parameters

dataSource

An `IEnumerator` collection to use as the data source.

fieldNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

This method takes a collection of values and an array of field names. The two must contain the same number of elements. Each name in the array of field names must be the same as the corresponding merge field name in the template.

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the main template document (the part of the document that is not within a repeat block) may not exceed the number of values in the data source defined by SetDataSource. However, the number of values in the data source may be greater than the number of merge fields in the main template document.

Examples

C#

```
wordTempl.SetDataSource(collValues, arrNames);
```

vb.net

```
wordTempl.SetDataSource(collValues, arrNames)
```

WordTemplate.SetDataSource(System.Data.DataSet, String)

Description

Sets a main document data source to an ADO.NET DataSet. A DataSet contains a collection of DataTables. If you set a main document data source to a DataSet, WordWriter will get the first row of the first DataTable in the DataSet.

Signature: C#

```
public void SetDataSource(System.Data.DataSet ds, System.String name)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal ds As System.Data.DataSet, ByVal name As String)
```

Parameters

ds

An ADO.NET DataSet to use as the template's data source. WordWriter will use the first row of the first DataTable in the DataSet as the data source.

name

Specifies the data source name for the set of merge fields.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in a set (merge fields that share the same data source name or number) may not exceed the number of values in the data source defined by SetDataSource. However, the number of values in the data source may be greater than the number of merge fields in the set.

This method may be called once for each set of merge fields in the main document.

Examples

C#

```
WordTemplate oWW = new WordTemplate();
          oWW.Open(@"c:\temp\template.doc");
          string strSQL = "SELECT LastName, FirstName, " +
               "TitleOfCourtesy, Address, City, Region, PostalCode " +
               "FROM Employees WHERE EmployeeID=1";

          //--- Create an OLEDB connection.
          OleDbConnection oConn = new OleDbConnection(strConnString);

          //--- Create an instance of the OleDbDataAdapter
          //--- object.
          OleDbDataAdapter oAdpt = new OleDbDataAdapter(strSQL, oConn);

          //--- Create a DataSet and fill it with the data
          //--- returned by the SQL database query.
          DataSet oDS = new DataSet();
          oAdpt.Fill(oDS);

          //--- Pass the DataSet to SetDataSource. WordWriter
          //--- will use the first DataTable in the DataSet as
          //--- the data source.
          oWW.SetDataSource(oDS, "Employee");
          ...
```

vb.net

```
Dim oWW As New WordTemplate()
          oWW.Open("c:\temp\template.doc")
          Dim strSQL As String = "SELECT LastName, FirstName, " & _
               "TitleOfCourtesy, Address, City, Region, PostalCode " & _
               "FROM Employees WHERE EmployeeID=1"

          '--- Create an OLEDB connection.
          Dim oConn As New OleDbConnection(strConnString)

          '--- Create an instance of the OleDbDataAdapter
          '--- object.
          Dim oAdpt As New OleDbDataAdapter(strSQL, oConn)

          '--- Create a DataSet and fill it with the data
          '--- returned by the SQL database query.
          Dim oDS As New DataSet()
          oAdpt.Fill(oDS)

          '--- Pass the DataSet to SetDataSource. WordWriter
          '--- will use the first DataTable in the DataSet as
          '--- the data source.
          oWW.SetDataSource(oDS, "Employee")
          ...
```

WordTemplate.SetDataSource(System.Data.DataSet)

Description

Sets a template's data source to an ADO.NET DataSet. A DataSet contains a collection of DataTables. If you set a template's data source to a DataSet, WordWriter will get the first row of the first DataTable in the DataSet.

Signature: C#

```
public void SetDataSource(System.Data.DataSet ds)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal ds As System.Data.DataSet)
```

Parameters

ds

An ADO.NET DataSet to use as the template's data source. WordWriter will use the first row of the first DataTable in the DataSet as the data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

Do not call this method more than once for a single instance of WordTemplate. To set multiple main document data sources, use the following methods:
SetDataSource(Object(), String(), String)
SetDataSource(System.Data.DataSet, String)
SetDataSource(System.Data.DataTable, String)
SetDataSource(System.Data.IDataReader, String)

Examples

C#

```
WordTemplate oWW = new WordTemplate();
          oWW.Open(@"c:\temp\template.doc");
          string strSQL = "SELECT LastName, FirstName, " +
               "TitleOfCourtesy, Address, City, Region, PostalCode " +
               "FROM Employees WHERE EmployeeID=1";

          //--- Create an OLEDB connection.
          OleDbConnection oConn = new OleDbConnection(strConnString);

          //--- Create an instance of the OleDbDataAdapter
          //--- object.
          OleDbDataAdapter oAdpt = new OleDbDataAdapter(strSQL, oConn);

          //--- Create a DataSet and fill it with the data
          //--- returned by the SQL database query.
          DataSet oDS = new DataSet();
          oAdpt.Fill(oDS);

          //--- Pass the DataSet to SetDataSource. WordWriter
          //--- will use the first DataTable in the DataSet as
          //--- the data source.
          oWW.SetDataSource(oDS);
```

vb.net

```
Dim oWW As New WordTemplate()
          oWW.Open("c:\temp\template.doc")
          Dim strSQL As String = "SELECT LastName, FirstName, " & _
               "TitleOfCourtesy, Address, City, Region, PostalCode " & _
               "FROM Employees WHERE EmployeeID=1"

          '--- Create an OLEDB connection.
          Dim oConn As New OleDbConnection(strConnString)

          '--- Create an instance of the OleDbDataAdapter
          '--- object.
          Dim oAdpt As New OleDbDataAdapter(strSQL, oConn)

          '--- Create a DataSet and fill it with the data
          '--- returned by the SQL database query.
          Dim oDS As New DataSet()
          oAdpt.Fill(oDS)

          '--- Pass the DataSet to SetDataSource. WordWriter
          '--- will use the first DataTable in the DataSet as
          '--- the data source.
          oWW.SetDataSource(oDS)
```

WordTemplate.SetDataSource(System.Data.DataTable, String)

Description

Sets a main document data source to an ADO.NET DataTable. If the specified DataTable contains more than one row, WordWriter will use the first row as the data source.

Signature: C#

```
public void SetDataSource(System.Data.DataTable dt, System.String name)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dt As System.Data.DataTable, ByVal name As String)
```

Parameters

dt

An ADO.NET DataTable to use as the data source. WordWriter will use the first row of the DataTable as the data source.

name

Specifies the data source name for the set of merge fields.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

This method may be called once for each set of merge fields in the main document.

Examples

C#

```
WordTemplate oWW = new WordTemplate();
          oWW.Open(@"c:\temp\template.doc");
          string strSQL = "SELECT LastName, FirstName, " +
               "TitleOfCourtesy, Address, City, Region, PostalCode " +
               "FROM Employees WHERE EmployeeID=1";

          //--- Create an OLEDB connection.
          OleDbConnection oConn = new OleDbConnection(strConnString);

          //--- Create an instance of the OleDbDataAdapter
          //--- object.
          OleDbDataAdapter oAdpt = new OleDbDataAdapter(strSQL, oConn);

          //--- Create a DataTable and fill it with the data
          //--- returned by the SQL database query.
          DataTable oDT = new DataTable();
          oAdpt.Fill(oDT);

          //--- Pass the DataTable to SetDataSource.
          oWW.SetDataSource(oDT, "Employee");
```

vb.net

```
Dim oWW As New WordTemplate()
          oWW.Open("c:\temp\template.doc")
          Dim strSQL As String = "SELECT LastName, FirstName, " & _
               "TitleOfCourtesy, Address, City, Region, PostalCode " & _
               "FROM Employees WHERE EmployeeID=1"

          '--- Create an OLEDB connection.
          Dim oConn As New OleDbConnection(strConnString)

          '--- Create an instance of the OleDbDataAdapter
          '--- object.
          Dim oAdpt As New OleDbDataAdapter(strSQL, oConn)

          '--- Create a DataTable and fill it with the data
          '--- returned by the SQL database query.
          Dim oDT As New DataTable()
          oAdpt.Fill(oDT)

          '--- Pass the DataTable to SetDataSource.
          oWW.SetDataSource(oDT, "Employee")
```

WordTemplate.SetDataSource(System.Data.DataTable)

Description

Sets a template's data source to an ADO.NET DataTable. If the specified DataTable contains more than one row, WordWriter will use the first row as the data source.

Signature: C#

```
public void SetDataSource(System.Data.DataTable dt)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dt As System.Data.DataTable)
```

Parameters

dt

An ADO.NET DataTable to use as the data source. WordWriter will use the first row of the DataTable as the data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

Do not call this method more than once for a single instance of WordTemplate. To set multiple main document data sources, use the following methods:
SetDataSource(Object(), String(), String)
SetDataSource(System.Data.DataSet, String)
SetDataSource(System.Data.DataTable, String)
SetDataSource(System.Data.IDataReader, String)

Examples

C#

```
WordTemplate oWW = new WordTemplate();
          oWW.Open(@"c:\temp\template.doc");
          string strSQL = "SELECT LastName, FirstName, TitleOfCourtesy, " +
               "Address, City, Region, PostalCode " +
               "FROM Employees WHERE EmployeeID=1";

          //--- Create an OLEDB connection.
          OleDbConnection oConn = new OleDbConnection(strConnString);

          //--- Create an instance of the OleDbDataAdapter
          //--- object.
          OleDbDataAdapter oAdpt = new OleDbDataAdapter(strSQL, oConn);

          //--- Create a DataTable and fill it with the data
          //--- returned by the SQL database query.
          DataTable oDT = new DataTable();
          oAdpt.Fill(oDT);

          //--- Pass the DataTable to SetDataSource.
          oWW.SetDataSource(oDT);
```

vb.net

```
Dim oWW As New WordTemplate()
          oWW.Open("c:\temp\template.doc")
          Dim strSQL As String = "SELECT LastName, FirstName, " & _
               "TitleOfCourtesy, Address, City, Region, PostalCode " & _
               "FROM Employees WHERE EmployeeID=1"

          '--- Create an OLEDB connection.
          Dim oConn As New OleDbConnection(strConnString)

          '--- Create an instance of the OleDbDataAdapter
          '--- object.
          Dim oAdpt As New OleDbDataAdapter(strSQL, oConn)

          '--- Create a DataTable and fill it with the data
          '--- returned by the SQL database query.
          Dim oDT As New DataTable()
          oAdpt.Fill(oDT)

          '--- Pass the DataTable to SetDataSource.
          oWW.SetDataSource(oDT)
```

WordTemplate.SetDataSource(System.Data.IDataReader, String)

Description

Sets a main document data source to an an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader, or an AdomdDataReader. If the specified DataReader returns more than one row, WordWriter will use the first row as the data source.

Signature: C#

```
public void SetDataSource(System.Data.IDataReader dr, System.String name)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dr As System.Data.IDataReader, ByVal name As String)
```

Parameters

dr

A SqlDataReader, OleDbDataReader or an AdomdDataReader to use as the data source. WordWriter will use the first row of the DataReader as the data source.

name

Specifies the data source name for the set of merge fields.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

This method may be called once for each set of merge fields in the main document.

Examples

C#

```
WordTemplate oWW = new WordTemplate();
          oWW.Open(@"c:\temp\template.doc");
          string strSQL = "SELECT LastName, FirstName, " +
               "TitleOfCourtesy, Address, City, Region, PostalCode " +
               "FROM Employees WHERE EmployeeID=1";

          //--- Create an OLEDB connection.
          OleDbConnection oConn = new OleDbConnection(strConnString);

          //--- Create an OleDbCommand
          OleDbCommand oCmd = new OleDbCommand(strSQL, oConn);

          //--- Get an OleDbDataReader
          oConn.Open();
          OleDbDataReader oReader =
               oCmd.ExecuteReader(CommandBehavior.CloseConnection);

          //--- Pass the DataReader to SetDataSource.
          oWW.SetDataSource(oReader, "Employee");

          //--- Close the reader (should do this in a finally block)
          oReader.Close();
          ...
```

vb.net

```
Dim oWW As New WordTemplate()
          oWW.Open("c:\temp\template.doc")
          Dim strSQL As String = "SELECT LastName, FirstName, " & _
               "TitleOfCourtesy, Address, City, Region, PostalCode " & _
               "FROM Employees WHERE EmployeeID=1"

          '--- Create an OLEDB connection.
          Dim oConn As New OleDbConnection(strConnString)

          '--- Create an OleDbCommand
          Dim oCmd As New OleDbCommand(strSQL, oConn)

          '--- Get an OleDbDataReader
          oConn.Open()
          Dim oReader As OleDbDataReader = _
               oCmd.ExecuteReader(CommandBehavior.CloseConnection)

          '--- Pass the DataReader to SetDataSource.
          oWW.SetDataSource(oReader, "Employee")

          '--- Close the reader (should do this in a finally block)
          oReader.Close()
          ...
```

WordTemplate.SetDataSource(System.Data.IDataReader)

Description

Sets a template's data source to an an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader or an AdomdDataReader. If the specified DataReader returns more than one row, WordWriter will use the first row as the data source.

Signature: C#

```
public void SetDataSource(System.Data.IDataReader dr)
```

Signature: VB.NET

```
Public Sub SetDataSource(ByVal dr As System.Data.IDataReader)
```

Parameters

dr

A SqlDataReader, OleDbDataReader or an AdomdDataReader to use as the data source. WordWriter will use the first row of the DataReader as the data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

Do not call this method more than once for a single instance of WordTemplate. To set multiple main document data sources, use the following methods:
SetDataSource(Object(), String(), String)
SetDataSource(System.Data.DataSet, String)
SetDataSource(System.Data.DataTable, String)
SetDataSource(System.Data.IDataReader, String)

Examples

C#

```
WordTemplate oWW = new WordTemplate();
          oWW.Open(@"c:\temp\template.doc");
          string strSQL = "SELECT LastName, FirstName, " +
               "TitleOfCourtesy, Address, City, Region, PostalCode " +
               "FROM Employees WHERE EmployeeID=1";

          //--- Create an OLEDB connection.
          OleDbConnection oConn = new OleDbConnection(strConnString);

          //--- Create an OleDbCommand
          OleDbCommand oCmd = new OleDbCommand(strSQL, oConn);

          //--- Get an OleDbDataReader
          oConn.Open();
          OleDbDataReader oReader =
               oCmd.ExecuteReader(CommandBehavior.CloseConnection);

          //--- Pass the DataReader to SetDataSource.
          oWW.SetDataSource(oReader);

          //--- Close the reader (should do this in a finally block)
          oReader.Close();
          ...
```

vb.net

```
Dim oWW As New WordTemplate()
          oWW.Open("c:\temp\template.doc")
          Dim strSQL As String = "SELECT LastName, FirstName, " & _
               "TitleOfCourtesy, Address, City, Region, PostalCode " & _
               "FROM Employees WHERE EmployeeID=1"

          '--- Create an OLEDB connection.
          Dim oConn As New OleDbConnection(strConnString)

          '--- Create an OleDbCommand
          Dim oCmd As New OleDbCommand(strSQL, oConn)

          '--- Get an OleDbDataReader
          oConn.Open()
          Dim oReader As OleDbDataReader = _
               oCmd.ExecuteReader(CommandBehavior.CloseConnection)

          '--- Pass the DataReader to SetDataSource.
          oWW.SetDataSource(oReader)

          '--- Close the reader (should do this in a finally block)
          oReader.Close()
          ...
```

WordTemplate.SetMailMerge

Overload	Description
SetMailMerge(Object(,), String())	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(Object(,), String(), Int32, Boolean)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(Object(), String())	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(Object()(), String())	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(Object()(), String(), Int32, Boolean)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

SetMailMerge(System.Collections.ICollection)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the entire page as a repeat block, adding a new page when needed. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(System.Collections.IEnumerator)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.
SetMailMerge(System.Data.DataSet)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.DataSet, Int32)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.DataTable)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.DataTable, Int32)	Sets the MailMerge's data source to a DataTable. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.IDataReader)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(System.Data.IDataReader, Int32)	Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

SharePoint Integration Extension Methods

Introduced in build 4.5

Overload	Description
SetMailMerge(Microsoft.SharePoint.SPList, Int32)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(Microsoft.SharePoint.SPList)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

SetMailMerge(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, Int32)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

WordTemplate.SetMailMerge(Object(,), String(), Int32, Boolean)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

Signature: C#

```
public void SetMailMerge(System.Object[,] table, System.String[] columnNames, int maxRows, boolean
transpose)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal table As Object(,), ByVal columnNames As String(), ByVal maxRows As
Integer, ByVal transpose As Boolean)
```

Parameters

table

The two-dimensional (rectangular) array of objects to use as the data source.

columnNames

The one-dimensional array of strings that represent the field names to be replaced by the data. These names must match the column names from your data source.

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant WordTemplate.ALL_ROWS .

transpose

If transpose is set to `true`, the array is treated as Object[column, row]. If transpose is set to `false`, the array is treated as Object[row, column].

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
...
          //--- A 2-D rectangular array of values
          //--- This is a "transposed" array with columns in the first
          //--- dimension and rows in the second
          object[,] data = new object[,]{
               {"Knoxville", "Boston",
               "Washington", "Seattle",
                "Chicago", "New York",
               "Atlanta", "Los Angeles",
               "Houston"},
               {"Tennessee", "Massachusetts",
               "DC", "Washington",
               "Illinois", "New York",
               "Georgia", "California",
               "Texas"}
               };

          //--- Names array, elements correspond to merge field names
          string[] names = new string[]{"City","State"};
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the 2-D rectangular data array
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          //--- Transpose is set to true to handle the transposed array
          wt.SetMailMerge(data, names, WordTemplate.ALL_ROWS, true);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
...
          '--- A 2-D rectangular array of values
          '--- This is a "transposed" array with columns in the first
          '--- dimension and rows in the second
          Dim data(,) As Object = { _
               {"Knoxville", "Boston", _
               "Washington", "Seattle", _
               "Chicago", "New York", _
               "Atlanta", "Los Angeles", _
               "Houston"}, _
               {"Tennessee", "Massachusetts", _
               "DC", "Washington", _
               "Illinois", "New York", _
               "Georgia", "California", _
               "Texas"} _
               }

          '--- names array, elements correspond to merge field names
          Dim names() As String = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the 2-D rectangular data array
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          '--- Transpose is set to true to handle the transposed array
          wt.SetMailMerge(data, names, WordTemplate.ALL_ROWS, True)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
```

WordTemplate.SetMailMerge(Object(,), String())

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

Signature: C#

```
public void SetMailMerge(System.Object[,] table, System.String[] columnNames)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal table As Object(,), ByVal columnNames As String())
```

Parameters

table

The two-dimensional(rectangular) array of objects to use as the data source.

columnNames

The one-dimensional array of strings that represent the field names to be replaced by the data. These names must match the column names from your data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
...
          //--- A 2-D rectangular array of values
          object[,] data = new object[,]{
               {"Knoxville","Tennessee"},
               {"Boston","Massachusetts"},
               {"Washington","DC"},
               {"Seattle","Washington"},
               {"Chicago", "Illinois"},
               {"New York","New York"},
               {"Atlanta", "Georgia"},
               {"Los Angeles", "California"},
               {"Houston", "Texas"}
          };

          //--- names array, elements correspond to merge field names
          string[] names = new string[]{"City","State"};
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the 2-D rectangular data array
          wt.SetMailMerge(data, names);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
...
          '--- A 2-D rectangular array of values
          Dim data(,) As Object = { _
               {"Knoxville", "Tennessee"}, _
               {"Boston", "Massachusetts"}, _
               {"Washington", "DC"}, _
               {"Seattle", "Washington"}, _
               {"Chicago", "Illinois"}, _
               {"New York", "New York"}, _
               {"Atlanta", "Georgia"}, _
               {"Los Angeles", "California"}, _
               {"Houston", "Texas"} _
          }

          '--- names array, elements correspond to merge field names
          Dim names() As String = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the 2-D rectangular data array
          wt.SetMailMerge(data, names)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
          ...
```

WordTemplate.SetMailMerge(Object(), String())

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

Signature: C#

```
public void SetMailMerge(System.Object[] data, System.String[] fieldNames)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal data As Object(), ByVal fieldNames As String())
```

Parameters

data

The one-dimensional array of objects to use as the data source.

fieldNames

The one-dimensional array of strings that represent the field names to be replaced by the data. These names must match the column names from your data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
...
          //--- A one-dimensional array of values
          object[] data = new object[]{"Knoxville","Tennessee"};

          //--- Names array, elements correspond to merge field names
          string[] names = new string[]{"City","State"};
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the one-dimensional data array
          wt.SetMailMerge(data, names);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
...
          '--- A one-dimensional arrray of values
          Dim data() As Object = {"Knoxville", "Tennessee"}

          '--- names array, elements correspond to merge field names
          Dim names() As String = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the one-dimensional array
          wt.SetMailMerge(data, names)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
          ...
```

WordTemplate.SetMailMerge(Object()(), String(), Int32, Boolean)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

Signature: C#

```
public void SetMailMerge(System.Object[][] jaggedArray, System.String[] columnNames, int maxRows,
boolean transpose)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal jaggedArray As Object()(), ByVal columnNames As String(), ByVal maxRows
As Integer, ByVal transpose As Boolean)
```

Parameters

jaggedArray

The jagged array (array-of-arrays) to use as the data source.

columnNames

The one-dimensional array of strings that represent the field names to be replaced by the data. These names must match the column names from your data source.

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant WordTemplate.ALL_ROWS.

transpose

If transpose is set to `true`, the array is treated as Object[column][row]. If transpose is set to `false`, the array is treated as Object[row][column].

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
...
          //--- A 2-D jagged array of values
          //--- This is a "transposed" array with columns in the first
          //--- dimension and rows in the second
          object[][] data = new object[][]{
               new string[]{"Knoxville", "Boston",
               "Washington", "Seattle",
               "Chicago", "New York",
               "Atlanta", "Los Angeles",
               "Houston"},
               new string[]{"Tennessee", "Massachusetts",
               "DC", "Washington",
               "Illinois", "New York",
               "Georgia", "California",
               "Texas"}
               };

          //--- Names array, elements correspond to merge field names
          string[] names = new string[] { "City", "State" };
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the 2-D jagged data array
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          //--- Transpose is set to true to handle the transposed array
          wt.SetMailMerge(data, names, WordTemplate.ALL_ROWS, true);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
'--- A 2-D jagged array of values
          '--- This is a "transposed" array with columns in the first
          '--- dimension and rows in the second
          Dim data()() As Object = { _
               New Object() {"Knoxville", "Boston", _
              "Washington", "Seattle", _
               "Chicago", "New York", _
               "Atlanta", "Los Angeles", _
               "Houston"}, _
               New Object() {"Tennessee", "Massachusetts", _
               "DC", "Washington", _
               "Illinois", "New York", _
               "Georgia", "California", _
               "Texas"} _
               }

          '--- names array, elements correspond to merge field names
          Dim names() As String = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the jagged data array
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          '--- Transpose is set to true to handle the transposed array
          wt.SetMailMerge(data, names, WordTemplate.ALL_ROWS, True)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
```

WordTemplate.SetMailMerge(Object()(), String())

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

Signature: C#

```
public void SetMailMerge(System.Object[][] jaggedArray, System.String[] columnNames)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal jaggedArray As Object()(), ByVal columnNames As String())
```

Parameters

jaggedArray

The jagged array (array-of-arrays) to use as the data source.

columnNames

The one-dimensional array of strings that represent the field names to be replaced by the data. These names must match the column names from your data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
...
          //--- A 2-D jagged array of values
          object[][] data = new object[][]{
               new string[]{"Knoxville","Tennessee"},
               new string[]{"Boston","Massachusetts"},
               new string[]{"Washington","DC"},
               new string[]{"Seattle","Washington"},
               new string[]{"Chicago", "Illinois"},
               new string[]{"New York","New York"},
               new string[]{"Atlanta", "Georgia"},
               new string[]{"Los Angeles", "California"},
               new string[]{"Houston", "Texas"}
               };

          //--- Names array, elements correspond to merge field names
          string[] names = new string[]{"City", "State"};
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the 2-D jagged data array
          wt.SetMailMerge(data, names);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
...
          '--- A 2-D jagged array of values
          Dim data()() As Object = { _
               New Object() {"Knoxville", "Tennessee"}, _
               New Object() {"Boston", "Massachusetts"}, _
               New Object() {"Washington", "DC"}, _
               New Object() {"Seattle", "Washington"}, _
               New Object() {"Chicago", "Illinois"}, _
               New Object() {"New York", "New York"}, _
               New Object() {"Atlanta", "Georgia"}, _
               New Object() {"Los Angeles", "California"}, _
               New Object() {"Houston", "Texas"} _
               }

          '--- names array, elements correspond to merge field names
          Dim names() As String = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the 2-D jagged data array
          wt.SetMailMerge(data, names)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
          ...
```

WordTemplate.SetMailMerge(System.Collections.ICollection)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the entire page as a repeat block, adding a new page when needed. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

Signature: C#

```
public void SetMailMerge(System.Collections.ICollection dataSource)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal dataSource As System.Collections.ICollection)
```

Parameters

dataSource

An ICollection to use as the data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Word 2003 binary template file type (.doc/.dot), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
wt.SetMailMerge(coll);
```

vb.net

```
wt.SetMailMerge(coll)
```

WordTemplate.SetMailMerge(System.Collections.IEnumerator)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields. This should largely imitate the behavior of a Microsoft Word Mail merge.

Signature: C#

```
public void SetMailMerge(System.Collections.IEnumerator dataSource)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal dataSource As System.Collections.IEnumerator)
```

Parameters

dataSource

An IEnumerator collection to use as a data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
wt.SetMailMerge(coll);
```

vb.net

```
wt.SetMailMerge(coll)
```

WordTemplate.SetMailMerge(System.Data.DataSet, Int32)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(System.Data.DataSet ds, int maxRows)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal ds As System.Data.DataSet, ByVal maxRows As Integer)
```

Parameters

ds

An ADO.NET DataSet to use as a data source.

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
//--- Create connection string for Excel file data
          string connString =
               "Provider=Microsoft.Jet.OLEDB.4.0;" +
               "Data Source=" + Server.MapPath("data/MailMergeData.xls") + ";" +
               "Extended Properties=Excel 8.0;";

          //--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          OleDbConnection conn = new OleDbConnection(connString);
          conn.Open();
          OleDbCommand cmd = new OleDbCommand("SELECT * FROM [Sheet1$]", conn);
          OleDbDataAdapter adapter = new OleDbDataAdapter();
          adapter.SelectCommand = cmd;

          //--- Dump the data into a DataSet
          DataSet ds = new DataSet();
          adapter.Fill(ds);
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the DataSet
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(ds, WordTemplate.ALL_ROWS);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
'--- Create connection string for Excel file data
          Dim connString As String = _
               "Provider=Microsoft.Jet.OLEDB.4.0;" amp; _
               "Data Source=" & Server.MapPath("data/MailMergeData.xls") & ";" & _
               "Extended Properties=Excel 8.0;"

          '--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          Dim conn As New OleDbConnection(connString)
          conn.Open()
          Dim cmd As New OleDbCommand("SELECT * FROM [Sheet1$]", conn)
          Dim adapter As New OleDbDataAdapter()
          adapter.SelectCommand = cmd

          '--- Dump the data into a DataSet
          Dim ds As New DataSet()
          adapter.Fill(ds)
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template\MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the DataSet
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(ds, WordTemplate.ALL_ROWS)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
```

WordTemplate.SetMailMerge(System.Data.DataSet)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(System.Data.DataSet ds)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal ds As System.Data.DataSet)
```

Parameters

table

An ADO.NET DataSet to use as a data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
//--- Create connection string for Excel file data
          string connString =
               "Provider=Microsoft.Jet.OLEDB.4.0;" +
               "Data Source=" + Server.MapPath("data/MailMergeData.xls") + ";" +
               "Extended Properties=Excel 8.0;";

          //--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          OleDbConnection conn = new OleDbConnection(connString);
          conn.Open();
          OleDbCommand cmd = new OleDbCommand("SELECT * FROM [Sheet1$]", conn);
          OleDbDataAdapter adapter = new OleDbDataAdapter();
          adapter.SelectCommand = cmd;

          //--- Dump the data into a DataSet
          DataSet ds = new DataSet();
          adapter.Fill(ds);
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the DataSet
          wt.SetMailMerge(ds);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
'--- Create connection string for Excel file data
          Dim connString As String = _
               "Provider=Microsoft.Jet.OLEDB.4.0;" & _
               "Data Source=" & Server.MapPath("data/MailMergeData.xls") + ";" & _
               "Extended Properties=Excel 8.0;"

          '--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          Dim conn As New OleDbConnection(connString)
          conn.Open()
          Dim cmd As New OleDbCommand("SELECT * FROM [Sheet1$]", conn)
          Dim adapter As New OleDbDataAdapter()
          adapter.SelectCommand = cmd

          '--- Dump the data into a DataSet
          Dim ds As New DataSet()
          adapter.Fill(ds)
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template\MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the DataSet
          wt.SetMailMerge(ds)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
```

WordTemplate.SetMailMerge(System.Data.DataTable, Int32)

Description

Sets the MailMerge's data source to a DataTable. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(System.Data.DataTable dt, int maxRows)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal dt As System.Data.DataTable, ByVal maxRows As Integer)
```

Parameters

dt

An ADO.NET DataTable to use as a data source.

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
//--- Create connection string for Excel file data
          string connString =
               "Provider=Microsoft.Jet.OLEDB.4.0;" +
               "Data Source=" + Server.MapPath("data/MailMergeData.xls") + ";" +
               "Extended Properties=Excel 8.0;";

          //--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          OleDbConnection conn = new OleDbConnection(connString);
          conn.Open();
          OleDbCommand cmd = new OleDbCommand("SELECT * FROM [Sheet1$]", conn);
          OleDbDataAdapter adapter = new OleDbDataAdapter();
          adapter.SelectCommand = cmd;

          //--- Dump the data into a DataTable
          DataTable dt = new DataTable();
          adapter.Fill(dt);
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the DataTable
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(dt, WordTemplate.ALL_ROWS);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
'--- Create connection string for Excel file data
          Dim connString As String = _
               "Provider=Microsoft.Jet.OLEDB.4.0;" & _
               "Data Source=" & Server.MapPath("data/MailMergeData.xls") & ";" & _
               "Extended Properties=Excel 8.0;"

          '--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          Dim conn As New OleDbConnection(connString)
          conn.Open()
          Dim cmd As New OleDbCommand("SELECT * FROM [Sheet1$]", conn)
          Dim adapter As New OleDbDataAdapter()
          adapter.SelectCommand = cmd

          '--- Dump the data into a DataTable
          Dim dt As New DataTable()
          adapter.Fill(dt)
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template\MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the DataTable
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(dt, WordTemplate.ALL_ROWS)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
```

WordTemplate.SetMailMerge(System.Data.DataTable)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(System.Data.DataTable dt)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal dt As System.Data.DataTable)
```

Parameters

dt

An ADO.NET DataTable to use as a data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
//--- Create connection string for Excel file data
          string connString =
               "Provider=Microsoft.Jet.OLEDB.4.0;" +
               "Data Source=" + Server.MapPath("data/MailMergeData.xls") + ";" +
               "Extended Properties=Excel 8.0;";

          //--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          OleDbConnection conn = new OleDbConnection(connString);
          conn.Open();
          OleDbCommand cmd = new OleDbCommand("SELECT * FROM [Sheet1$]", conn);
          OleDbDataAdapter adapter = new OleDbDataAdapter();
          adapter.SelectCommand = cmd;

          //--- Dump the data into a DataTable
          DataTable dt = new DataTable();
          adapter.Fill(dt);
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the DataTable
          wt.SetMailMerge(dt);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
          ...
```

vb.net

```
'--- Create connection string for Excel file data
          Dim connString As String = _
               "Provider=Microsoft.Jet.OLEDB.4.0;" & _
               "Data Source=" & Server.MapPath("data/MailMergeData.xls") & ";" & _
               "Extended Properties=Excel 8.0;"

          '--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          Dim conn As New OleDbConnection(connString)
          conn.Open()
          Dim cmd As New OleDbCommand("SELECT * FROM [Sheet1$]", conn)
          Dim adapter As New OleDbDataAdapter()
          adapter.SelectCommand = cmd

          '--- Dump the data into a DataTable
          Dim dt As New DataTable()
          adapter.Fill(dt)
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template\MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the DataTable
          wt.SetMailMerge(dt)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
```

WordTemplate.SetMailMerge(System.Data.IDataReader, Int32)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the

NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(System.Data.IDataReader dr, int maxRows)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal dr As System.Data.IDataReader, ByVal maxRows As Integer)
```

Parameters

dr

A SqlDataReader, OleDbDataReader, or AdomdDataReader to use as the data source.

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
//--- Create connection string for Excel file data
          string connString =
               "Provider=Microsoft.Jet.OLEDB.4.0;" +
               "Data Source=" + Server.MapPath("data/MailMergeData.xls") + ";" +
               "Extended Properties=Excel 8.0;";

          //--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          //--- Place all of the data in a DataReader object
          OleDbConnection conn = new OleDbConnection(connString);
          conn.Open();
          OleDbCommand cmd = new OleDbCommand("SELECT * FROM [Sheet1$]", conn);
          OleDbDataReader dr = cmd.ExecuteReader();
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the DataReader
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(dr, WordTemplate.ALL_ROWS);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
'--- Create connection string for Excel file data
          Dim connString As String = _
               "Provider=Microsoft.Jet.OLEDB.4.0;" & _
               "Data Source=" & Server.MapPath("data/MailMergeData.xls") & ";" & _
               "Extended Properties=Excel 8.0;"

          '--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          '--- Place all of the data in a DataReader object
          Dim conn As New OleDbConnection(connString)
          conn.Open()
          Dim cmd As New OleDbCommand("SELECT * FROM [Sheet1$]", conn)
          Dim dr As OleDbDataReader = cmd.ExecuteReader()
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template\MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the DataReader
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(dr, WordTemplate.ALL_ROWS)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
```

WordTemplate.SetMailMerge(System.Data.IDataReader)

Description

Sets the MailMerge's data source to a two-dimensional (rectangular) array of objects. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(System.Data.IDataReader dr)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal dr As System.Data.IDataReader)
```

Parameters

dr

A SqlDataReader, OleDbDataReader, or AdomdDataReader to use as the data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetMailMerge include field names (**«fieldname»**) and field ordinals (**«#1»**).

Examples

C#

```
//--- Create connection string for Excel file data
          string connString =
               "Provider=Microsoft.Jet.OLEDB.4.0;" +
               "Data Source=" + Server.MapPath("data/MailMergeData.xls") + ";" +
               "Extended Properties=Excel 8.0;";

          //--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          //--- Place all of the data in a DataReader object
          OleDbConnection conn = new OleDbConnection(connString);
          conn.Open();
          OleDbCommand cmd = new OleDbCommand("SELECT * FROM [Sheet1$]", conn);
          OleDbDataReader dr = cmd.ExecuteReader();
          WordTemplate wt = new WordTemplate();
          wt.Open(Server.MapPath("template/MergeFieldTest.doc"));

          //--- Set the mail merge
          //--- The data source is the DataReader
          wt.SetMailMerge(dr);
          wt.Process();
          wt.Save(Page.Response, "output.doc", false);
```

vb.net

```
'--- Create connection string for Excel file data
          Dim connString As String = _
               "Provider=Microsoft.Jet.OLEDB.4.0;" & _
               "Data Source=" & Server.MapPath("data/MailMergeData.xls") & ";" & _
               "Extended Properties=Excel 8.0;"

          '--- Use OleDb to get all data from Sheet1 of the MailMergeData.xls spreadsheet
          '--- Place all of the data in a DataReader object
          Dim conn As New OleDbConnection(connString)
          conn.Open()
          Dim cmd As New OleDbCommand("SELECT * FROM [Sheet1$]", conn)
          Dim dr As OleDbDataReader = cmd.ExecuteReader()
          Dim wt As New WordTemplate()
          wt.Open(Server.MapPath("template\MergeFieldTest.doc"))

          '--- Set the mail merge
          '--- The data source is the DataReader
          wt.SetMailMerge(dr)
          wt.Process()
          wt.Save(Page.Response, "output.doc", False)
```

WordTemplate.SetRepeatBlock

Overload	Description
SetRepeatBlock(System.Data.DataSet, String)	Sets the specified repeat block's data source to an ADO.NET DataSet. A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.
SetRepeatBlock(System.Data.DataSet, String, Int32)	Sets the specified repeat block's data source to an ADO.NET DataSet.
SetRepeatBlock(System.Data.DataTable, String)	Sets the specified repeat block's data source to an ADO.NET DataTable.
SetRepeatBlock(System.Data.DataTable, String, Int32)	Sets the specified repeat block's data source to an ADO.NET DataTable.
SetRepeatBlock(System.Data.IDataReader, String)	Sets the specified repeat block's data source to an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader or an AdomdDataReader.

SetRepeatBlock(System.Data.IDataReader, String, Int32)	Sets the specified repeat block's data source to an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader, or an AdomdDataReader.
SetRepeatBlock(Object(), String(), String)	Sets the specified repeat block's data source to a 1-dimensional array of objects.
SetRepeatBlock(Object(,), String(), String)	Sets the specified repeat block's data source to a multidimensional (rectangular) array of objects.
SetRepeatBlock(Object(,), String(), String, Int32, Boolean)	Sets the specified repeat block's data source to a multidimensional (rectangular) array of objects.
SetRepeatBlock(Object()(), String(), String)	Sets the specified repeat block's data source to a jagged array (array-of-arrays) of objects.
SetRepeatBlock(Object()(), String(), String, Int32, Boolean)	Sets the specified repeat block's data source to a jagged array (array-of-arrays) of objects.
SetRepeatBlock(System.Collections.IEnumerator, String)	Sets the specified repeat block's data source to an `IEnumerator` collection.
SetRepeatBlock(System.Collections.IEnumerator, String, String)	Sets the specified repeat block's data source to an `IEnumerator` collection.
SetRepeatBlock(System.Collections.ICollection, String)	Sets the specified repeat block's data source to an `ICollection`.
SetRepeatBlock(System.Collections.ICollection, String, String)	Sets the specified repeat block's data source to an `ICollection`.

SharePoint Integration Extension Methods

Introduced in build 4.5

Overload	Description
SetRepeatBlock(Microsoft.SharePoint.SPList, String, Int32)	Sets the specified repeat block's data source to a SharePoint List.
SetRepeatBlock(Microsoft.SharePoint.SPList, String)	Sets the specified repeat block's data source to a SharePoint List.
SetRepeatBlock(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, Int32)	Sets the specified repeat block's data source to a SharePoint View.
SetRepeatBlock(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)	Sets the specified repeat block's data source to a SharePoint View.

WordTemplate.SetRepeatBlock(Object(,), String(), String, Int32, Boolean)

Description

Sets the specified repeat block's data source to a multidimensional (rectangular) array of objects.

Signature: C#

```
public void SetRepeatBlock(System.Object[,] table, System.String[] columnNames, System.String
bookmark, int maxRows, boolean transpose)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal table As Object(,), ByVal columnNames As String(), ByVal bookmark As
String, ByVal maxRows As Integer, ByVal transpose As Boolean)
```

Parameters

table

An object array of values to use as a data source. WordWriter will insert these values in the repeat block's merge fields. Objects in the array must not be arrays themselves. Objects in the array can be null. Each object in the data source array must have a corresponding field name in the array of field names. By default, the first dimension corresponds to row and the second to column (that is, Object[row,column]).

columnNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

maxRows

Specifies the maximum number of repetitions to write to the file. If Next fields are enabled (see EnableNEXTFields), then the number of rows imported from the data source is `maxRows*(numberOfNextFields+1)`. If Next fields are not enabled the number of rows imported is equal to the number of repetitions written out. To write the maximum rows available, use the constant WordTemplate.ALL_ROWS.

transpose

If transpose is set to `true`, the array is treated as Object[column][row]. If transpose is set to `false`, the array is treated as Object[row][column].

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty data source to `SetRepeatBlock` WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- A 2-D rectangular array of values
          //--- This is a "transposed" array with columns in the first
          //--- dimension and rows in the second
          string[,] ValuesArr = {{"Boston", "Miami", "Merchantville"},
               {"MA", "FL", "NJ"}};

          //--- Names array, elements correspond to merge field names
          string[] NamesArr = {"City", "State"};
          WordTemplate wt = new WordTemplate();
          wt.Open("template.doc");

          //--- Set the repeat block defined by the bookmark "Block"
          //--- The data source is the 2-D rectangular ValuesArr array
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          //--- Transpose is true to handle the transposed array
          wt.SetRepeatBlock(ValuesArr, NamesArr, "Block",
               WordTemplate.ALL_ROWS, true);
          wt.Process();
          wt.Save("out.doc");
```

vb.net

```
'--- A 2-D rectangular array of values
          '--- This is a "transposed" array with columns in the first
          '--- dimension and rows in the second
          Dim ValuesArr As String(,) = {{"Boston", "Miami", "Merchantville"}, _
               {"MA", "FL", "NJ"}}

          '--- Names array, elements correspond to merge field names
          Dim NamesArr As String() = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open("template.doc")

          '--- Set the repeat block defined by the bookmark "Block"
          '--- The data source is the 2-D rectangular ValuesArr array
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          '--- Transpose is true to handle the transposed array
          wt.SetRepeatBlock(ValuesArr, NamesArr, "Block", _
               WordTemplate.ALL_ROWS, True)
          wt.Process()
          wt.Save("out.doc")
```

WordTemplate.SetRepeatBlock(Object(,), String(), String)

Description

Sets the specified repeat block's data source to a multidimensional (rectangular) array of objects.

Signature: C#

```
public void SetRepeatBlock(System.Object[,] table, System.String[] columnNames, System.String
bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal table As Object(,), ByVal columnNames As String(), ByVal bookmark As
String)
```

Parameters

table

An object array of values to use as a data source. WordWriter will insert these values in the repeat block's merge fields. Objects in the array must not be arrays themselves. Objects in the array can be null. Each object in the data source array must have a corresponding field name in the array of field names.

columnNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty data source to `SetRepeatBlock` WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- A 2-D rectangular array of values
          string[,] ValuesArr = {{"Boston", "MA"},
               {"Miami", "FL"},
               {"Merchantville", "NJ"}};

          //--- Names array, elements correspond to merge field names
          string[] NamesArr = {"City", "State"};
          WordTemplate wt = new WordTemplate();
          wt.Open("template.doc");

          //--- Set the repeat block defined by the bookmark "Block"
          //--- The data source is the 2-D rectangular ValuesArr array
          wt.SetRepeatBlock(ValuesArr, NamesArr, "Block");
          wt.Process();
          wt.Save("out.doc");
```

vb.net

```
'--- A 2-D rectangular array of values
          Dim ValuesArr As String(,)  = {{"Boston", "MA"}, _
               {"Miami", "FL"}, _
               {"Merchantville", "NJ"}}

          '--- Names array, elements correspond to merge field names
          Dim NamesArr As String() = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open("template.doc")

          '--- Set the repeat block defined by the bookmark "Block"
          '--- The data source is the 2-D rectangular ValuesArr array
          wt.SetRepeatBlock(ValuesArr, NamesArr, "Block")
          wt.Process()
          wt.Save("out.doc")
```

WordTemplate.SetRepeatBlock(Object(), String(), String)

Description

Sets the specified repeat block's data source to a 1-dimensional array of objects.

Signature: C#

```
public void SetRepeatBlock(System.Object[] dataRow, System.String[] fieldNames, System.String
bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dataRow As Object(), ByVal fieldNames As String(), ByVal bookmark As
String)
```

Parameters

data

An object array of values to use as a data source. WordWriter will insert these values in the repeat block's merge fields. Objects in the array must not be arrays themselves. Objects in the array can be null. Each object in the data source array must have a corresponding field name in the array of field names.

fieldNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template. The array of data source values and the array of field names must contain the same number of elements.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty data source to `SetRepeatBlock` WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
string[] NamesArr = {"CompanyName", "ProductName", "URL"};
          object[] ValuesArr = {"SoftArtisans", "WordWriter",
               "http://www.softartisans.com"};
          WordTemplate WordTempl = new WordTemplate();
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"));

          //--- Set the data source of the repeat block "ProductInfo"
          //--- by passing the value array and the name array initialized above.
          WordTempl.SetRepeatBlock(ValuesArr, NamesArr, "ProductInfo");
          WordTempl.Process();
          WordTempl.Save(Page.Response, "BasicGenerated.doc", false);
```

vb.net

```
Dim NamesArr As String() = {"CompanyName", "ProductName", "URL"}
          Dim ValuesArr As Object() = {"SoftArtisans", "WordWriter", _
               "http://www.softartisans.com"}
          Dim WordTempl As new WordTemplate()
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"))

          '--- Set the data source of the repeat block "ProductInfo"
          '--- by passing the value array and the name array initialized above.
          WordTempl.SetRepeatBlock(ValuesArr, NamesArr, "ProductInfo")
          WordTempl.Process()
          WordTempl.Save(Page.Response, "BasicGenerated.doc", False)
```

WordTemplate.SetRepeatBlock(Object()(), String(), String, Int32, Boolean)

Description

Sets the specified repeat block's data source to a jagged array (array-of-arrays) of objects.

Signature: C#

```
public void SetRepeatBlock(System.Object[][] jaggedArray, System.String[] columnNames, System.String
bookmark, int maxRows, boolean transpose)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal jaggedArray As Object()(), ByVal columnNames As String(), ByVal
bookmark As String, ByVal maxRows As Integer, ByVal transpose As Boolean)
```

Parameters

jaggedArray

An jagged array of objects to use as a data source. the first dimension corresponds to row and the second to column (that is, Object[row][column]). WordWriter will insert these values in the repeat block's merge fields.

columnNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

maxRows

Specifies the maximum number of repetitions to write to the file. If Next fields are enabled (see EnableNEXTFields), then the number of rows imported from the data source is `maxRows*(numberOfNextFields+1)`. If Next fields are not enabled the number of rows imported is equal to the number of repetitions written out. To write the maximum rows available, use the constant WordTemplate.ALL_ROWS.

transpose

If transpose is set to `true`, the array is treated as Object[column][row]. If transpose is set to `false`, the array is treated as Object[row][column].

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty data source to `SetRepeatBlock` WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- A 2-D jagged array of values
          //--- This is a "transposed" array with columns in the first
          //--- dimension and rows in the second
          string[][] ValuesArr = new string[][] {
               new string[] {"Boston", "Miami", "Merchantville"},
               new string[] {"MA", "FL", "NJ"}
               };

          //--- Names array, elements correspond to merge field names
          string[] NamesArr = {"City", "State"};
          WordTemplate wt = new WordTemplate();
          wt.Open("template.doc");

          //--- Set the repeat block defined by the bookmark "Block"
          //--- The data source is the 2-D rectangular ValuesArr array
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          //--- Transpose is true to handle the transposed array
          wt.SetRepeatBlock(ValuesArr, NamesArr, "Block",
               WordTemplate.ALL_ROWS, true);
          wt.Process();
          wt.Save("out.doc");
```

vb.net

```
'--- A 2-D jagged array of values
          '--- This is a "transposed" array with columns in the first
          '--- dimension and rows in the second
          Dim ValuesArr()() As String = { _
               New String() {"Boston", "Miami", "Merchantville"}, _
               New String(){"MA", "FL", "NJ"} _
               }

          '--- Names array, elements correspond to merge field names
          Dim NamesArr As String() = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open("template.doc")

          '--- Set the repeat block defined by the bookmark "Block"
          '--- The data source is the 2-D rectangular ValuesArr array
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          '--- Transpose is true to handle the transposed array
          wt.SetRepeatBlock(ValuesArr, NamesArr, "Block", _
               WordTemplate.ALL_ROWS, True)
          wt.Process()
          wt.Save("out.doc")
```

WordTemplate.SetRepeatBlock(Object()(), String(), String)

Description

Sets the specified repeat block's data source to a jagged array (array-of-arrays) of objects.

Signature: C#

```
public void SetRepeatBlock(System.Object[][] jaggedArray, System.String[] columnNames, System.String
bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal jaggedArray As Object()(), ByVal columnNames As String(), ByVal
bookmark As String)
```

Parameters

jaggedArray

An jagged array of objects to use as a data source. the first dimension corresponds to row and the second to column (that is, Object[row][column]). WordWriter will insert these values in the repeat block's merge fields.

columnNames

A string array of data source field names. These must be the same as the corresponding merge field names in the template.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty data source to `SetRepeatBlock` WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- A 2-D jagged array of values
          string[][] ValuesArr = new string[][] {
               new string[] {"Boston", "MA"},
               new string[] {"Miami", "FL"},
               new string[] {"Merchantville", "NJ"}
               };

          //--- Names array, elements correspond to merge field names
          string[] NamesArr = {"City", "State"};
          WordTemplate wt = new WordTemplate();
          wt.Open("template.doc");

          //--- Set the repeat block defined by the bookmark "Block"
          //--- The data source is the 2-D rectangular ValuesArr array
          wt.SetRepeatBlock(ValuesArr, NamesArr, "Block");
          wt.Process();
          wt.Save("out.doc");
```

vb.net

```
'--- A 2-D jagged array of values
          Dim ValuesArr()() As String = { _
               New String() {"Boston", "MA"}, _
               New String(){"Miami", "FL"}, _
               New String() {"Merchantville", "NJ"} _
               }

          '--- Names array, elements correspond to merge field names
          Dim NamesArr As String() = {"City", "State"}
          Dim wt As New WordTemplate()
          wt.Open("template.doc")

          '--- Set the repeat block defined by the bookmark "Block"
          '--- The data source is the 2-D rectangular ValuesArr array
          wt.SetRepeatBlock(ValuesArr, NamesArr, "Block")
          wt.Process()
          wt.Save("out.doc")
```

WordTemplate.SetRepeatBlock(System.Collections.ICollection, String, String)

Description

Sets the specified repeat block's data source to an `ICollection`.

Signature: C#

```
public void SetRepeatBlock(System.Collections.ICollection dataSource, System.String fieldName,
System.String bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dataSource As System.Collections.ICollection, ByVal fieldName As
String, ByVal bookmark As String)
```

Parameters

dataSource

An `ICollection` to use as the data source.

field

The name of the merge field at which to insert the values.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter bookmark must exist in the template. All merge fields in the repeat block will bind to the specified data source.

Examples

C#

```
wTempl.SetRepeatBlock(coll, "ProductID", "Products");
```

vb.net

```
wTempl.SetRepeatBlock(coll, "ProductID", "Products")
```

WordTemplate.SetRepeatBlock(System.Collections.ICollection, String)

Description

Sets the specified repeat block's data source to an `ICollection`.

Signature: C#

```
public void SetRepeatBlock(System.Collections.ICollection dataSource, System.String bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dataSource As System.Collections.ICollection, ByVal bookmark As
String)
```

Parameters

dataSource

An `ICollection` to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.

5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter bookmark must exist in the template. All merge fields in the repeat block will bind to the specified data source.

Examples

C#

```
wTempl.SetRepeatBlock(coll, "Products");
```

vb.net

```
wTempl.SetRepeatBlock(coll, "Products")
```

WordTemplate.SetRepeatBlock(System.Collections.IEnumerator, String, String)

Description

Sets the specified repeat block's data source to an `IEnumerator` collection.

Signature: C#

```
public void SetRepeatBlock(System.Collections.IEnumerator dataSource, System.String fieldName,
System.String bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dataSource As System.Collections.IEnumerator, ByVal fieldName As
String, ByVal bookmark As String)
```

Parameters

dataSource

An `IEnumerator` collection to use as the data source.

field

The name of the merge field at which to insert the values.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter bookmark must exist in the template. All merge fields in the repeat block will bind to the specified data source.

Examples

C#

```
wTempl.SetRepeatBlock(coll, "ProductID", "Products");
```

vb.net

```
wTempl.SetRepeatBlock(coll, "ProductID", "Products")
```

WordTemplate.SetRepeatBlock(System.Collections.IEnumerator, String)

Description

Sets the specified repeat block's data source to an `IEnumerator` collection.

Signature: C#

```
public void SetRepeatBlock(System.Collections.IEnumerator dataSource, System.String bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dataSource As System.Collections.IEnumerator, ByVal bookmark As
String)
```

Parameters

dataSource

An `IEnumerator` collection to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter bookmark must exist in the template. All merge fields in the repeat block will bind to the specified data source.

Examples

C#
`wTempl.SetRepeatBlock(coll, "Products");`

vb.net
`wTempl.SetRepeatBlock(coll, "Products")`

WordTemplate.SetRepeatBlock(System.Data.DataSet, String, Int32)

Description

Sets the specified repeat block's data source to an ADO.NET DataSet.

Signature: C#

```
public void SetRepeatBlock(System.Data.DataSet ds, System.String bookmark, int maxRows)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal ds As System.Data.DataSet, ByVal bookmark As String, ByVal maxRows As
Integer)
```

Parameters

ds

The DataSet to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

maxRows

Specifies the maximum number of repetitions to write to the file. If Next fields are enabled (see EnableNEXTFields), then the number of rows imported from the data source is `maxRows*(numberOfNextFields+1)`. If Next fields are not enabled the number of rows imported is equal to the number of repetitions written out. To write the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty DataSet to `SetRepeatBlock` - by either setting `maxRows` to 0 or passing an empty DataSet - WordWriter will not include

the specified repeat block in the generated Word file.

Examples

C#

```
//--- A dataset with the data to be repeated
          DataSet ds = GetDataSet();
          WordTemplate oWW = new WordTemplate();
          oWW.Open("c:\\template.doc");

          //--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "ds" is the DataSet containing the data to be
          //--- imported and repeated in the repeat block
          //--- The first DataTable in the DataSet will be used
          //--- Set MaxRows to 10.  No matter how many DataRows
          //--- are in the DataSet, only 10 will be imported
          oWW.SetRepeatBlock(ds, "Page", 10);
          oWW.Process();
          oWW.Save(Page.Response, "Output.doc", false);
```

vb.net

```
'--- A dataset with the data to be repeated
          Dim ds As DataSet = GetDataSet()
          Dim oWW As New WordTemplate()
          oWW.Open("c:\template.doc")

          '--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "ds" is the DataSet containing the data to be
          '--- imported and repeated in the repeat block
          '--- The first DataTable in the DataSet will be used
          '--- Set MaxRows to 10.  No matter how many DataRows
          '--- are in the DataSet, only 10 will be imported
          oWW.SetRepeatBlock(ds, "Page", 10)
          oWW.Process()
          oWW.Save(Page.Response, "Output.doc", False)
```

WordTemplate.SetRepeatBlock(System.Data.DataSet, String)

Description

Sets the specified repeat block's data source to an ADO.NET DataSet. A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

Signature: C#

```
public void SetRepeatBlock(System.Data.DataSet ds, System.String bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal ds As System.Data.DataSet, ByVal bookmark As String)
```

Parameters

ds

The DataSet to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty DataSet to `SetRepeatBlock`, WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- A dataset with the data to be repeated
          DataSet ds = GetDataSet();
          WordTemplate oWW = new WordTemplate();
          oWW.Open("c:\\template.doc");

          //--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "ds" is the DataSet containing the data to be
          //--- imported and repeated in the repeat block
          //--- The first DataTable in the DataSet will be used
          oWW.SetRepeatBlock(ds, "Page");
          oWW.Process();
          oWW.Save(Page.Response, "Output.doc", false);
```

vb.net

```
'--- A dataset with the data to be repeated
          Dim ds As DataSet = GetDataSet()
          Dim oWW As New WordTemplate()
          oWW.Open("c:\template.doc")

          '--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "ds" is the DataSet containing the data to be
          '--- imported and repeated in the repeat block
          '--- The first DataTable in the DataSet will be used
          oWW.SetRepeatBlock(ds, "Page")
          oWW.Process()
          oWW.Save(Page.Response, "Output.doc", False)
```

WordTemplate.SetRepeatBlock(System.Data.DataTable, String, Int32)

Description

Sets the specified repeat block's data source to an ADO.NET DataTable.

Signature: C#

```
public void SetRepeatBlock(System.Data.DataTable dt, System.String bookmark, int maxRows)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dt As System.Data.DataTable, ByVal bookmark As String, ByVal maxRows
As Integer)
```

Parameters

dt

The DataTable to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

maxRows

Specifies the maximum number of repetitions to write to the file. If Next fields are enabled (see EnableNEXTFields), then the number of rows imported from the data source is `maxRows*(numberOfNextFields+1)`. If Next fields are not enabled the number of rows imported is equal to the number of repetitions written out. To write the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty DataTable to `SetRepeatBlock` - by either setting `maxRows` to 0 or passing an empty DataTable - WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- A DataTable with the data to be repeated
          DataTable ds = GetDataTable();
          WordTemplate oWW = new WordTemplate();
          oWW.Open("c:\\template.doc");

          //--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "dt" is the DataTable containing the data to be
          //--- imported and repeated in the repeat block
          //--- Set MaxRows to 10.  Only 10 rows from the DataTable
          //--- will be imported
          oWW.SetRepeatBlock(dt, "Page", 10);
          oWW.Process();
          oWW.Save(Page.Response, "Output.doc", false);
```

vb.net

```
'--- A DataTable with the data to be repeated
          Dim ds As DataTable = GetDataTable()
          Dim oWW As New WordTemplate()
          oWW.Open("c:\template.doc")

          '--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "dt" is the DataTable containing the data to be
          '--- imported and repeated in the repeat block
          '--- Set MaxRows to 10.  Only 10 rows from the DataTable
          '--- will be imported
          oWW.SetRepeatBlock(dt, "Page", 10)
          oWW.Process()
          oWW.Save(Page.Response, "Output.doc", False)
```

WordTemplate.SetRepeatBlock(System.Data.DataTable, String)

Description

Sets the specified repeat block's data source to an ADO.NET DataTable.

Signature: C#

```
public void SetRepeatBlock(System.Data.DataTable dt, System.String bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dt As System.Data.DataTable, ByVal bookmark As String)
```

Parameters

dt

The DataTable to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty DataTable to `SetRepeatBlock`, WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- A DataTable with the data to be repeated
          DataTable ds = GetDataTable();
          WordTemplate oWW = new WordTemplate();
          oWW.Open("c:\\template.doc");

          //--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "dt" is the DataTable containing the data to be
          //--- imported and repeated in the repeat block
          oWW.SetRepeatBlock(dt, "Page");
          oWW.Process();
          oWW.Save(Page.Response, "Output.doc", false);
```

vb.net

```
'--- A DataTable with the data to be repeated
          Dim ds As DataTable = GetDataTable()
          Dim oWW As New WordTemplate()
          oWW.Open("c:\template.doc")

          '--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "dt" is the DataTable containing the data to be
          '--- imported and repeated in the repeat block
          oWW.SetRepeatBlock(dt, "Page")
          oWW.Process()
          oWW.Save(Page.Response, "Output.doc", False)
```

WordTemplate.SetRepeatBlock(System.Data.IDataReader, String, Int32)

Description

Sets the specified repeat block's data source to an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader, or an AdomdDataReader.

Signature: C#

```
public void SetRepeatBlock(System.Data.IDataReader dr, System.String bookmark, int maxRows)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dr As System.Data.IDataReader, ByVal bookmark As String, ByVal maxRows
As Integer)
```

Parameters

dr

a SqlDataReader, OleDbDataReader or and AdomdDataReader to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

maxRows

Specifies the maximum number of repetitions to write to the file. If Next fields are enabled (see EnableNEXTFields), then the number of rows

imported from the data source is `maxRows*(numberOfNextFields+1)`. If Next fields are not enabled the number of rows imported is equal to the number of repetitions written out. To write the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty DataReader to `SetRepeatBlock` - by either setting `maxRows` to 0 or passing an empty DataReader - WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- Get a DataReader
          //--- WordWriter accepts OleDbDataReader, SqlDataReader, and AdomdDataReader
          OleDbDataReader dr = GetOleDbDataReader();
          WordTempalte oWW = new WordTemplate();
          oWW.Open(Page.MapPath("./template.doc"));

          //--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "dr" is the DataReader containing information to be
          //--- imported and repeated in the repeat block
          //--- Set MaxRows to 10.  Only 10 rows from the DataReader
          //--- will be imported
          oWW.SetRepeatBlock(dr, "Page", 10);
          oWW.Process();
          oWW.Save(Page.Response, "EmployeeOutput.doc", false);

          //--- Close the DataReader in a finally block
          dr.Close();

          private OleDbDataReader GetOleDbDataReader()
          {
               OleDbConnection Conn = new OleDbConnection(connString);

               //--- SQL Query for categories.
               string CategorySQL = "SELECT CategoryID, CategoryName," +
                    "Description FROM Categories";
               Conn.Open();
               return new OleDbCommand(CategorySQL, Conn).ExecuteReader(
                    CommandBehavior.CloseConnection);
          }
```

vb.net

```
'--- Get a DataReader
          '--- WordWriter accepts OleDbDataReader, SqlDataReader, and AdomdDataReader
          Dim dr As OleDbDataReader = GetOleDbDataReader()
          Dim oWW As New WordTemplate()
          oWW.Open(Page.MapPath("./template.doc"))

          '--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "dr" is the DataReader containing information to be
          '--- imported and repeated in the repeat block
          '--- Set MaxRows to 10.  Only 10 rows from the DataReader
          '--- will be imported
          oWW.SetRepeatBlock(dr, "Page", 10)
          oWW.Process()
          oWW.Save(Page.Response, "EmployeeOutput.doc", False)

          '--- Close the DataReader in a finally block
          dr.Close()

          Private Function GetOleDbDataReader() As OleDbDataReader
               Dim Conn As New OleDbConnection(connString)

               '--- SQL Query for categories.
               Dim CategorySQL As String = "SELECT CategoryID, CategoryName," & _
                    "Description FROM Categories"
               Conn.Open()
               Return New OleDbCommand(CategorySQL, Conn).ExecuteReader( _
                    CommandBehavior.CloseConnection)
          End Function
```

WordTemplate.SetRepeatBlock(System.Data.IDataReader, String)

Description

Sets the specified repeat block's data source to an IDataReader interface, which may be either a SqlDataReader, OleDbDataReader or an AdomdDataReader.

Signature: C#

```
public void SetRepeatBlock(System.Data.IDataReader dr, System.String bookmark)
```

Signature: VB.NET

```
Public Sub SetRepeatBlock(ByVal dr As System.Data.IDataReader, ByVal bookmark As String)
```

Parameters

dr

a SqlDataReader, OleDbDataReader, or an AdomdDataReader to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

If you pass an empty DataReader to `SetRepeatBlock` WordWriter will not include the specified repeat block in the generated Word file.

Examples

C#

```
//--- Get a DataReader
          //--- WordWriter accepts OleDbDataReader, SqlDataReader, and AdomdDataReader
          OleDbDataReader dr = GetOleDbDataReader();
          WordTempalte oWW = new WordTemplate();
          oWW.Open(Page.MapPath("./template.doc"));

          //--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "dr" is the DataReader containing information to be
          //--- imported and repeated in the repeat block
          oWW.SetRepeatBlock(dr, "Page");
          oWW.Process();
          oWW.Save(Page.Response, "EmployeeOutput.doc", false);

          //--- Close the DataReader in a finally block
          dr.Close();

          private OleDbDataReader GetOleDbDataReader()
          {
               OleDbConnection Conn = new OleDbConnection(connString);

               //--- SQL Query for categories.
               string CategorySQL = "SELECT CategoryID, CategoryName," +
                    "Description FROM Categories";
               Conn.Open();
               return new OleDbCommand(CategorySQL, Conn).ExecuteReader(
                    CommandBehavior.CloseConnection);
          }
```

vb.net

```
'--- Get a DataReader
          '--- WordWriter accepts OleDbDataReader, SqlDataReader, and AdomdDataReader
          Dim dr As OleDbDataReader = GetOleDbDataReader()
          Dim oWW As New WordTemplate()
          oWW.Open(Page.MapPath("./template.doc"))

          '--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "dr" is the DataReader containing information to be
          '--- imported and repeated in the repeat block
          oWW.SetRepeatBlock(dr, "Page")
          oWW.Process()
          oWW.Save(Page.Response, "EmployeeOutput.doc", False)

          '--- Close the DataReader in a finally block
          dr.Close()

          Private Function GetOleDbDataReader() As OleDbDataReader
               Dim Conn As New OleDbConnection(connString)

               '--- SQL Query for categories.
               Dim CategorySQL As String = "SELECT CategoryID, CategoryName," & _
                    "Description FROM Categories"
               Conn.Open()
               Return New OleDbCommand(CategorySQL, Conn).ExecuteReader( _
                    CommandBehavior.CloseConnection)
          End Function
```

WordTemplate.UpdateIfFields

Description

Gets or sets whether IF fields should be evaluated server-side in the WordTemplate.

Signature: C#

```
public boolean UpdateIfFields{ get; set; }
```

Signature: VB.NET

```
Public Property UpdateIfFields() As Boolean
```

Remarks

If set to `true`, all IF fields on the template will be evaluated on the server side by the WordTemplate object. If set to `false`, the IF fields will not be evaluated without user action.

Examples

C#

```
WordTemplate WordTempl = new WordTemplate();
          WordTempl.UpdateIfFields = true;
```

vb.net

```
Dim WordTempl As New WordTemplate()
          WordTempl.UpdateIfFields = True
```

WordTemplate.Version

Description

Returns the exact version of WordWriter.

Signature: C#

```
public System.String Version{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Version() As String
```

Remarks

The format of the version string returned is:

```
SoftArtisans.ProductName Major.Minor.Patch.Build
            (mmddyyyy-hhmmss-buildMachineId)
```

Place Holder	Description
ProductName	A product name like "SoftArtisans.WordWriter." Note: the product name may be different depending on what license key is in effect.
Major	Major release number.
Minor	Minor release number.
Patch	Patch or bug fix number.
Build	Build number.
mmddyyyy	Date the build was compiled.
hhmmss	Time the build was compiled.
buildMachineId	Unique computer ID for the computer on which this product was compiled.

Examples

C#

```
WordTemplate WordTempl = new WordTemplate();
          string Version = WordTempl.Version;
          string[] NamesArr = {"WordWriterVersion", "DateTime"};
          object[] ValuesArr = {Version, DateTime.Now};
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"));
          WordTempl.SetDataSource(ValuesArr, NamesArr);
          WordTempl.Process();
          WordTempl.Save(Page.Response, "BasicGenerated.doc", false);
```

vb.net

```
Dim WordTempl As New WordTemplate()
          Dim Version As String = WordTempl.Version
          Dim NamesArr As String() = {"WordWriterVersion", "DateTime"}
          Dim ValuesArr As Object() = {Version, DateTime.Now}
          WordTempl.Open(Page.MapPath("./BasicTemplate.doc"))
          WordTempl.SetDataSource(ValuesArr, NamesArr)
          WordTempl.Process()
          WordTempl.Save(Page.Response, "BasicGenerated.doc", False)
```

WordTemplate.WordTemplate()

Description

Creates a new WordTemplate object.

Signature: C#

```
public WordTemplate ( )
```

Signature: VB.NET

```
Public Sub New WordTemplate ( )
```

Examples

C#

```
WordTemplate wt = new WordTemplate();
```

vb.net

```
Dim wlt As New WordTemplate()
```

WordApplication Programmer's Reference

- AbsolutePositioning — This class is used to specify the position of tables and paragraphs in a Word document. Although the term "absolute" is used, the position is relative to some location in the document or page. Absolute positioning is set in twips. 1 twip = 1/20 pt = 1/1440 in.
- AutoFormatter — When you import data to a table in your document, you can use the AutoFormatter interface to specify how the imported data should be formatted in the table. `AutoFormatter` contains a single method - FormatCell - which must be implemented in a class that you will use as an AutoFormatter. When you import data, set DataImportProperties.AutoFormatter to the class that you created.
- Bookmark — This class is used to represent a bookmark in the document. Bookmarks can be added to a document in Word or programmatically using WordWriter. They can only be retrieved form an existing document.
- Border — This class is used to specify border settings for various entities in a Word document.
- CharacterRun — The CharacterRun class represents a contiguous run of characters in the document, all of which have the same formatting.
- CompatibilitySettings — This class contains various settings so imported documents from other versions of Word and competing products will look the way they were intended to.
- DataImportProperties — This class is used to control settings for importing data into a table or table cell.
- Document — Represents a Word document.
- DocumentProperties — This class contains all file properties that are used by Word and Windows Explorer to display and store metadata about documents. None of these properties will affect the content of a document.
- DocumentProtection — This class is used to get and set document protection settings for a document.
- Element — This class is used to represent a region of the document that can be edited. This is the main class for manipulating the Word object model. Most elements of a Document inherit from the Element class.
- Field — This class is a low level representation of a field. This encompasses many objects in Word, such as merge fields, hyperlinks, TOC entries, etc. Fields are the generic form of what are contained in a Element's inline elements.
- Font — The Font class is used to specify formatting that should be applied to a character run. It also represents the fonts of named styles in the document.
- Hyperlink — This class represents a hyperlink field in a document.
- InlineImage — This class is used to represent an inline image in a Word document. The types of images that Word supports are jpg, png, bmp, and gif. Inline images are part of the *InlineElement* objects of an Element.
- List — This class is used to represent a list in a Word document. Lists are part of the *BlockElement* objects of an Element. Using this class, Lists can be read and written. Lists contain list entries, which are basically paragraphs with additional features.
- ListEntry — This class is used to represent an entry in a List inside a Word document. A list entry in Word is basically a numbered paragraph.
- ListLevel — This class is used to manipulate settings for a particular indent level. Things such as number and bullet text and indent change with each indent level in Word.
- MergeField — This class represents a merge field in a document. Merge fields are used in WordWriter templates as data placeholders.
- NamedStyle — The NamedStyle class represents a named style, such as "Normal", "Heading 1", etc., that is embedded in a Word document.
- Paragraph — Represents a paragraph in the Word document. Technically, a paragraph is a block of text that ends with a carriage return.
- ParagraphFormatting — The ParagraphFormatting class is used to specify formatting that should be applied to a paragraph. It also represents the paragraph formatting of named styles in the document.

- Position — This class is used to represent a cursor position in the document. It sits between two characters and its position is represented as the index of the character it sits behind. A Position object can be obtained from an Element.
- SearchMatch — A SearchMatch object represents a match in the document found by using the method Element.Search.
- Section — This class is used to represent a section of the document. Many documents will only contain one section. More complex documents will divide the document into multiple sections. A section can be roughly correlated to a chapter in a book. Each section can have different formatting settings, including:
- Shading — This class is represents background shading for text, table cells, etc.
- Styles — This class is used to store the styles of a document. It is essentially a collection of named styles.
- Table — Represents a table in the Word document.
- TableCell — Represents a table cell in a table in the Word document.
- TableFormatting — The TableFormatting class is used to specify formatting that should be applied to a table. It also represents the table formatting of some named styles in the document.
- TwipsConverter — Twips are the units of measurement used by the WordWriter API. A single twip represents 1/1440 of an inch. TwipsConverter is a utility class that converts between twips and other units of measurement.
- Units — The different types of units used by WordWriter
- Watermark — A Watermark object represents a document's watermark, which is an image or text that when printed appears in the background of a document. The property Document.Watermark returns a `Watermark` object.
- WordApplication — WordApplication is the main class for pure code-based Word document generation (for information about template-based workbook generation, see WordTemplate). This class is an engine used to open, create, and write (save to disk or stream to a browser) Word documents. A single instance of `WordApplication` can generate multiple documents.
- WordChangeTracking — This class is used to get and set Word change tracking settings for a document.
- WordHyphenSettings — This class is used to get and set Word hyphen settings for a document.
- WordNumberFormat — This class is used to enumerate the set of Word's built-in number formats Word. It is used for setting how List elemements, footnote numbers, and endnote numbers will appear.
- WordViewSettings — This class is used manipulate the View settings for a document.
- WordWriterException — WordWriterException is thrown by the Element.Search() and Element.SearchAndReplace() methods.

AbsolutePositioning

Description

This class is used to specify the position of tables and paragraphs in a Word document. Although the term "absolute" is used, the position is relative to some location in the document or page. Absolute positioning is set in twips. 1 twip = 1/20 pt = 1/1440 in.

Signature: C#

```
public sealed class AbsolutePositioning
```

Signature: VB.NET

```
Public NotInheritable Class AbsolutePositioning
```

Remarks

You cannot create an AbsolutePositioning object using the `new` keyword. You must get it from a ParagraphFormatting or Table object using the `AbsolutePositioning` property.

The following sample gets the `AbsolutePositioning` object of the first table of a document.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Table firstTable = (Table) doc.GetElements(Element.Type.Table)[0];
          AbsolutePositioning ap = firstTable.AbsolutePositioning;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim firstTable As Table = doc.GetElements(Element.Type.Table)(0)
          Dim ap As AbsolutePositioning = firstTable.AbsolutePositioning
```

Properties

Name	Description
Height	Sets or returns an `int` that represents the height in twips.
HeightExact	Specifies whether AbsolutePositioning.Height is an exact height (true) or a minimum height (false).
OriginX	Returns the horizontal origin of the position's AbsolutePositioning.X. Possible values are: Column, Margin, Page.
OriginY	Sets or returns the vertical origin of the position's AbsolutePositioning.Y. Possible values are: Margin, Page, Text.
Width	Sets or returns an `int` that represents the absolute width in twips. One twip = (1/20 pt) or (1/1440 in)
X	Sets or returns an `int` that represents the x coordinate in twips. One twip = (1/20 pt) or (1/1440 in)
Y	Sets or returns an `int` that represents the position's y coordinate in twips. One twip = (1/20 pt) or (1/1440 in)

Nested Classes

Name	Description
HorizontalOrigin	An AbsolutePositioning.HorizontalOrigin value specifies the vertical origin of a table or paragraph position's y coordinate.
VerticalOrigin	An AbsolutePositioning.VerticalOrigin value specifies the vertical origin of a table or paragraph position's y coordinate.

AbsolutePositioning.Height

Description

Sets or returns an `int` that represents the height in twips.

Signature: C#

```
public int Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Integer
```

Remarks

The width of Paragraph must be between 0.01" (15 twips) and 22" (31680 twips). The width of a table must be between 0" (0 twips) and 22" (31680 twips). Values beyond these will be automatically adjusted.

Examples

C#

```
//--- Get Height
          int height = oAbsolutePositioning.Height;

          //--- Set Height to 3 inches
          oAbsolutePositioning.Height = 4320;
```

vb.net

```
'--- Get Height
          Dim height As Integer = oAbsolutePositioning.Height

          '--- Set Height to 3 inches
          oAbsolutePositioning.Height = 4320
```

AbsolutePositioning.HeightExact

Description

Specifies whether AbsolutePositioning.Height is an exact height (true) or a minimum height (false).

Signature: C#

```
public boolean HeightExact{ get; set; }
```

Signature: VB.NET

```
Public Property HeightExact() As Boolean
```

Examples

C#

```
//--- Get HeightExact
          bool heightExact = oAbsolutePositioning.HeightExact;

          //--- Set HeightExact
          oAbsolutePositioning.Height = true;
```

vb.net

```
'--- Get HeightExact
          Dim heightExact As Boolean = oAbsolutePositioning.HeightExact

          '--- Set HeightExact
          oAbsolutePositioning.HeightExact = True
```

AbsolutePositioning.HorizontalOrigin

Description

An AbsolutePositioning.HorizontalOrigin value specifies the vertical origin of a table or paragraph position's y coordinate.

Signature: C#

```
public sealed class HorizontalOrigin : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class HorizontalOrigin
  Inherits TypesafeEnum
```

Values

Name	Description
Column	The position's x coordinate is relative to the left edge of the current column. If the page does not contain multiple columns, `HorizontalOrigin.Column` and `HorizontalOrigin.Margin` are equivalent.
Margin	The position's x coordinate is relative to the left margin of the page.
Page	The position's x coordinate is relative to the left border of the page.
Text	The position's x coordinate is relative to the position of the table or paragraph if it were not absolutely positioned.

AbsolutePositioning.OriginX

Description

Returns the horizontal origin of the position's AbsolutePositioning.X. Possible values are: Column, Margin, Page.

Signature: C#

```
public HorizontalOrigin OriginX{ get; set; }
```

Signature: VB.NET

```
Public Property OriginX() As HorizontalOrigin
```

Examples

C#

```
//--- Get HorizontalOrigin
          AbsolutePositioning.HorizontalOrigin oHorizontalOrigin = oAbsolutePositioning.OriginX;

          //--- Set HorizontalOrigin
          oAbsolutePositioning.OriginX = AbsolutePositioning.HorizontalOrigin.Margin;
```

vb.net

```
'--- Get HorizontalOrigin
          Dim oHorizontalOrigin As AbsolutePositioning.HorizontalOrigin = oAbsolutePositioning.OriginX

          '--- Set HorizontalOrigin
          oAbsolutePositioning.OriginX= AbsolutePositioning.HorizontalOrigin.Margin
```

AbsolutePositioning.OriginY

Description

Sets or returns the vertical origin of the position's AbsolutePositioning.Y. Possible values are: Margin, Page, Text.

Signature: C#

```
public VerticalOrigin OriginY{ get; set; }
```

Signature: VB.NET

```
Public Property OriginY() As VerticalOrigin
```

Examples

C#

```
//--- Get VerticalOrigin
          AbsolutePositioning.OriginY oVerticalOrigin = oAbsolutePositioning.VerticalOrigin;

          //--- Set VerticalOrigin
          oAbsolutePositioning.OriginY = AbsolutePositioning.VerticalOrigin.Page;
```

vb.net

```
'--- Get VerticalOrigin
          Dim oVerticalOrigin As AbsolutePositioning.OriginY = oAbsolutePositioning.VerticalOrigin

          '--- Set VerticalOrigin
          oAbsolutePositioning.OriginY = AbsolutePositioning.VerticalOrigin.Page
```

AbsolutePositioning.VerticalOrigin

Description

An AbsolutePositioning.VerticalOrigin value specifies the vertical origin of a table or paragraph position's y coordinate.

Signature: C#

```
public sealed class VerticalOrigin : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class VerticalOrigin
  Inherits TypesafeEnum
```

Values

Name	Description
Margin	The position's y coordinate is relative to the top margin of the page.
Page	The position's y coordinate is relative to the top border of the page.
Text	The position's y coordinate is relative to the position of the table or paragraph if it were not absolutely positioned.

AbsolutePositioning.Width

Description

Sets or returns an `int` that represents the absolute width in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Integer
```

Remarks

The width of Paragraph must be between 0.01" (15 twips) and 22" (31680 twips). The width of a table must be between 0" (0 twips) and 22" (31680 twips). Values beyond these will be automatically adjusted.

Examples

C#

```
//--- Get Width
          int width = oAbsolutePositioning.Width;

          //--- Set Width to 3 inches
          oAbsolutePositioning.Width = 4320;
```

vb.net

```
'--- Get Width
          Dim width As Integer = oAbsolutePositioning.Width

          '--- Set Width to 3 inches
          oAbsolutePositioning.Width = 4320
```

AbsolutePositioning.X

Description

Sets or returns an `int` that represents the x coordinate in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int X{ get; set; }
```

Signature: VB.NET

```
Public Property X() As Integer
```

Examples

C#

```
//--- Get X
          int x = oAbsolutePositioning.X;

          //--- Set X to 1 inch
          oAbsolutePositioning.X = 1440;
```

vb.net

```
'--- Get X
          Dim x As Integer = oAbsolutePositioning.X

          '--- Set X to 1 inch
          oAbsolutePositioning.X = 1440
```

AbsolutePositioning.Y

Description

Sets or returns an `int` that represents the position's y coordinate in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int Y{ get; set; }
```

Signature: VB.NET

```
Public Property Y() As Integer
```

Examples

C#

```
//--- Get Y
          int y = oAbsolutePositioning.Y;

          //--- Set X to 1 inch
          oAbsolutePositioning.Y = 1440;
```

vb.net

```
'--- Get Y
          Dim y As Integer = oAbsolutePositioning.Y

          '--- Set Y to 1 inch
          oAbsolutePositioning.Y = 1440
```

AutoFormatter

Description

When you import data to a table in your document, you can use the AutoFormatter interface to specify how the imported data should be formatted in the table. `AutoFormatter` contains a single method - FormatCell - which must be implemented in a class that you will use as an AutoFormatter. When you import data, set DataImportProperties.AutoFormatter to the class that you created.

Signature: C#

```
public abstract class AutoFormatter
```

Signature: VB.NET

```
Public MustInherit Class AutoFormatter
```

Methods

Name	Description
FormatCell(TableCell, Int32, Int32, AutoFormatter.Edge)	This method must be implemented by the class you want to use as an AutoFormatter. Then, when you import data, you can use DataImportProperties.AutoFormatter to set the AutoFormatter for that data import.

Nested Classes

Name	Description
Edge	An AutoFormatter.Edge value specifies the location of a table cell in relation to the outer edge of the table.

AutoFormatter.Edge

Description

An AutoFormatter.Edge value specifies the location of a table cell in relation to the outer edge of the table.

Signature: C#

```
public sealed class Edge : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Edge
  Inherits TypesafeEnum
```

Values

Name	Description
Bottom	The cell is in the bottom row of the table.
BottomLeft	The cell is in the bottom left corner of the table.
BottomRight	The cell is in the bottom right corner of the table.
None	The cell is not within a row or column at the outer edge of the table.
Left	The cell is in the left column of the table.
Right	The cell is in the right column of the table.
Top	The cell is in the top row of the table.
TopLeft	The cell is in the top left corner of the table.
TopRight	The cell is in the top right corner of the table.

AutoFormatter.FormatCell(TableCell, Int32, Int32, AutoFormatter.Edge)

Description

This method must be implemented by the class you want to use as an AutoFormatter. Then, when you import data, you can use DataImportProperties.AutoFormatter to set the AutoFormatter for that data import.

Signature: C#

```
public abstract void FormatCell(TableCell aCell, int aRow, int aColumn, Edge aLocation)
```

Signature: VB.NET

```
Public MustOverride Sub FormatCell(ByVal aCell As TableCell, ByVal aRow As Integer, ByVal aColumn As
Integer, ByVal aLocation As Edge)
```

Parameters

aCell

The table cell to format.

aRow

The 0-based row index of the cell to format.

aColumn

The 0-based column index of the cell to format.

aLocation

The location of the cell in relation to the outer edge of the table.

Bookmark

Description

This class is used to represent a bookmark in the document. Bookmarks can be added to a document in Word or programmatically using WordWriter. They can only be retrieved form an existing document.

Signature: C#

```
public sealed class Bookmark : Element
```

Signature: VB.NET

```
Public NotInheritable Class Bookmark
  Inherits Element
```

Remarks

To insert a bookmark into a document, use Element.CreateBookmark() or Table.CreateBookmarkOnRow(). To get a bookmark from an existing document use the Document.GetBookmark method.

The following sample shows you how to add a bookmark to a document and retrieve a bookmark from a document.

Examples

C#

```
//--- Create a bookmark
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          doc.CreateBookmark("MyBookmark");

          //--- Get an existing bookmark
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Bookmark bmark = doc.GetBookmark("ExistingBookmark");

          //--- Or, if you know the order of the bookmarks in the document
          //--- you can use the following:
          //Bookmark bmark = (Bookmark)doc.GetElements(Element.Type.Bookmark)[0];
```

vb.net

```
'--- Create a bookmark
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          doc.CreateBookmark("MyBookMark")

          '--- Get an existing bookmark
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim bmark As Bookmark = doc.GetBookmark("ExistingBookmark")

          '--- Or, if you know the order of the bookmarks in the document
          '--- you can use the following:
          'Dim bmark as Bookmark = doc.GetElements(Element.Type.Bookmark)(0)
```

Properties

Name	Description
Name	Returns a `String` representing the name of this bookmark.

Bookmark.Name

Description

Returns a `String` representing the name of this bookmark.

Signature: C#

```
public System.String Name{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Name() As String
```

Examples

C#

```
string bmarkName = bmark.Name;
```

vb.net

```
Dim bmarkName As String = bmark.Name
```

Border

Description

This class is used to specify border settings for various entities in a Word document.

Signature: C#

```
public sealed class Border
```

Signature: VB.NET

```
Public NotInheritable Class Border
```

Remarks

You cannot create a Border object using the `new` keyword. You must reference an existing Border using the `Border` property method of one of the following classes: Font, CharacterRun, Paragraph, ParagraphFormatting, Section, or TableCell.

The following sample gets the `Border` object of the upper left table cell of the first table in a document.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Border brdr =
               (Table)(doc.GetElements(Element.Type.Table)[0])[0, 0].Border;

          //--- Set Width to 1 point
          brdr.LineWidth = 8;
          brdr.Style = Border.LineStyle.Single;
```

vb.net

```
Dim app As WordApplication = New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim brdr As Border = _
               doc.GetElements(Element.Type.Table)(0)(0, 0).Border

          '--- Set Width to 1 point
          brdr.LineWidth = 8
          brdr.Style = Border.LineStyle.Single
```

Properties

Name	Description
Color	Sets or returns a `System.Drawing.Color` object that represents the color of this border.
HasShadow	Sets or returns a `boolean` that represents whether or not the border has a shadow effect.
LineWidth	Sets or returns an `int` that represents the line width of this border in 1/8 points. (The maximum width is 32 points.)
Padding	Sets or returns an `int` that represents the space to maintain between the border and the text in twips. One twip = (1/20 pt) or (1/1440 in)
Reverse	Sets or returns a boolean which specifies whether or not the border graphic is reversed.
Style	Sets or returns a Border.LineStyle object that represents the line style of this border. (Single, thick, double, etc.)

Nested Classes

Name	Description
LineStyle	Used to specify a line style for borders.
Location	Used to specify the border location. Under certain circumstance it is necessary to specify the location of a border. Not all of these will apply for all circumstances.

Border.Color

Description

Sets or returns a `System.Drawing.Color` object that represents the color of this border.

Signature: C#

```
public System.Drawing.Color Color{ get; set; }
```

Signature: VB.NET

```
Public Property Color() As System.Drawing.Color
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Borders tab > Color:

Examples

C#

```
//--- Get Color
          System.Drawing.Color borderColor = brdr.Color;

          //--- Set Color
          brdr.Color = System.Drawing.Color.Blue;
```

vb.net

```
'--- Get Color
          Dim borderColor As System.Drawing.Color = brdr.Color

          '--- Set Color
          brdr.Color = System.Drawing.Color.Blue
```

Border.HasShadow

Description

Sets or returns a `boolean` that represents whether or not the border has a shadow effect.

Signature: C#

```
public boolean HasShadow{ get; set; }
```

Signature: VB.NET

```
Public Property HasShadow() As Boolean
```

Remarks

This is set to false by default.

MS Word equivalent: Format menu > Borders and Shading... > Borders tab > Setting: > Shadow

Examples

C#

```
//--- Get HasShadow
          bool hasShadow = brdr.HasShadow;

          //--- Set HasShadow
          brdr.HasShadow = true;
```

vb.net

```
'--- Get HasShadow
          Dim hasShadow As Boolean = brdr.HasShadow

          '--- Set HasShadow
          brdr.HasShadow = True
```

Border.LineStyle

Description

Used to specify a line style for borders.

Signature: C#

```
public sealed class LineStyle : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class LineStyle
  Inherits TypesafeEnum
```

Values

Name	Description
DashDotStroked	Dash-dot-stroked
DashLargeGap	Dashed line with large gaps
DashSmallGap	Dashed line with small gaps between the dashes
Dot	Dotted line
DotDash	Dot-dash line
DotDotDash	Dot-dot-dash line
Double	Double line
DoubleWave	Two wavy lines
Emboss3d	Line with 3-dimensional embossed effect
Engrave3d	Line with 3-dimensional engraved effect
Hairline	Very thin line
None	No line
Single	Single line
Thick	Thick line
ThickThinLargeGap	Thick line over thin line with a large gap between the lines
ThickThinMediumGap	Thick line over thin line with a medium gap between the lines
ThickThinSmallGap	Thick line over thin line with a small gap between the lines
ThinThickLargeGap	Thin line over thick line with a large gap between the lines
ThinThickMediumGap	Thin line over thick line with a medium gap between the lines
ThinThickSmallGap	Thin line over thick line with a small gap between the lines
ThinThickThinLargeGap	Thick line with thin lines above and below, and large gaps between the lines
ThinThickThinMediumGap	Thick line with thin lines above and below, and medium gaps between the lines
ThinThickThinSmallGap	Thick line with thin lines above and below, and small gaps between the lines
Triple	Triple line
Unknown	
Wave	Wavy line

Border.LineWidth

Description

Sets or returns an `int` that represents the line width of this border in 1/8 points. (The maximum width is 32 points.)

Signature: C#

```
public int LineWidth{ get; set; }
```

Signature: VB.NET

```
Public Property LineWidth() As Integer
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Borders tab > Width:

Examples

C#

```
//--- Get LineWidth
          int lineWidth = brdr.LineWidth;

          //--- Set LineWidth to 1 point
          brdr.LineWidth = 8;
```

vb.net

```
'--- Get LineWidth
          Dim lineWidth As Integer = brdr.LineWidth

          '--- Set LineWidth to 1 point
          brdr.LineWidth = 8
```

Border.Location

Description

Used to specify the border location. Under certain circumstance it is necessary to specify the location of a border. Not all of these will apply for all circumstances.

Signature: C#

```
public sealed class Location : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Location
  Inherits TypesafeEnum
```

Values

Name	Description
Bar	A border that is placed outside of text when facing pages are displayed
Between	Border between conforming paragraphs. Conforming paragraphs share the following characteristics: • Both have the same AbsolutePositioning attributes. • Both have borders. • The widths of both paragraphs are equal. • Both paragraphs are within a table or not.
Bottom	Bottom border
Horizontal	All horizontal borders

Left	Left border
Right	Right border
Top	Top border
Vertical	All vertical borders

Border.Padding

Description

Sets or returns an `int` that represents the space to maintain between the border and the text in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int Padding{ get; set; }
```

Signature: VB.NET

```
Public Property Padding() As Integer
```

Remarks

The minimum width is 0 points and the maximum width is 32 points. Values beyond these will be automatically adjusted.

MS Word equivalent: Format menu > Borders and Shading... > Borders tab > Options button > From text

Examples

C#

```
//--- Get Padding
          int padding = brdr.Padding;

          //--- Set Padding to 4 points
          brdr.Padding = 80;
```

vb.net

```
'--- Get Padding
          Dim padding As Integer = brdr.Padding

          '--- Set Padding to 4 points
          brdr.Padding = 80
```

Border.Reverse

Description

Sets or returns a boolean which specifies whether or not the border graphic is reversed.

Signature: C#

```
public boolean Reverse{ get; set; }
```

Signature: VB.NET

```
Public Property Reverse() As Boolean
```

Remarks

This only has an effect on certain line styles which are not symmetrical (for example, Border.LineStyle.ThinThickSmallGap). Reversing the right and bottom graphics of certain line styles can create a "3D" effect. CharacterRuns have only one border. Paragraphs have different border parts. For borders around characterRuns, Reverse tells you whether or not the right and bottom sides of the border have reversed graphics. For borders around paragraphs, it is necessary to explicitly reverse the right and bottom sides.

Reverse defaults to false which means that by default borders will appear "boxed".

MS Word equivalent: Format menu > Borders and Shading... > Borders tab > Setting: > Box

Examples

C#

```
//--- Get Reverse
          bool reverse = brdr.Reverse;

          //--- Set a 3D border around a character run
          Border brdr = charRun.Font.Border;
          brdr.Reverse = true;
          brdr.LineWidth = 12;
          brdr.Style = Border.LineStyle.ThinThickSmallGap;

          //--- Set a 3D border around a paragraph
          para.Formatting.GetBorder(Border.Location.Top).Style = Border.LineStyle.ThinThickSmallGap;
          para.Formatting.GetBorder(Border.Location.Top).LineWidth = 12;
          para.Formatting.GetBorder(Border.Location.Left).Style = Border.LineStyle.ThinThickSmallGap;
          para.Formatting.GetBorder(Border.Location.Left).LineWidth = 12;
          para.Formatting.GetBorder(Border.Location.Bottom).Style =
Border.LineStyle.ThinThickSmallGap;
          para.Formatting.GetBorder(Border.Location.Bottom).LineWidth = 12;
          para.Formatting.GetBorder(Border.Location.Bottom).Reverse = true;
          para.Formatting.GetBorder(Border.Location.Right).Style = Border.LineStyle.ThinThickSmallGap;
          para.Formatting.GetBorder(Border.Location.Right).LineWidth = 12;
          para.Formatting.GetBorder(Border.Location.Right).Reverse = true;
```

vb.net

```
'--- Get Reverse
          Dim reverse As Boolean = brdr.Reverse

          '--- Set a 3D border around a character run
          Border brdr = charRun.Font.Border
          brdr.Reverse = True
          brdr.LineWidth = 12
          brdr.Style = Border.LineStyle.ThinThickSmallGap

          '--- Set a 3D border around a paragraph
          para.Formatting.GetBorder(Border.Location.Top).Style = Border.LineStyle.ThinThickSmallGap
          para.Formatting.GetBorder(Border.Location.Top).LineWidth = 12
          para.Formatting.GetBorder(Border.Location.Left).Style = Border.LineStyle.ThinThickSmallGap
          para.Formatting.GetBorder(Border.Location.Left).LineWidth = 12
          para.Formatting.GetBorder(Border.Location.Bottom).Style = Border.LineStyle.ThinThickSmallGap
          para.Formatting.GetBorder(Border.Location.Bottom).LineWidth = 12
          para.Formatting.GetBorder(Border.Location.Bottom).Reverse = True
          para.Formatting.GetBorder(Border.Location.Right).Style = Border.LineStyle.ThinThickSmallGap
          para.Formatting.GetBorder(Border.Location.Right).LineWidth = 12
          para.Formatting.GetBorder(Border.Location.Right).Reverse = True
```

Border.Style

Description

Sets or returns a Border.LineStyle object that represents the line style of this border. (Single, thick, double, etc.)

Signature: C#

```
public LineStyle Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As LineStyle
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Borders tab > Style:

Examples

C#

```
//--- Get LineStyle
          Border.LineStyle lnStyle = brdr.LineStyle;

          //--- Set LineStyle
          brdr.LineStyle = Border.LineStyle.Thick;
```

vb.net

```
'--- Get LineStyle
          Dim lnStyle As Border.LineStyle = brdr.LineStyle

          '--- Set LineStyle
          brdr.LineStyle = Border.LineStyle.Thick
```

CharacterRun

Description

The CharacterRun class represents a contiguous run of characters in the document, all of which have the same formatting.

Signature: C#

```
public sealed class CharacterRun : Element
```

Signature: VB.NET

```
Public NotInheritable Class CharacterRun
  Inherits Element
```

Remarks

To insert a new character run into a document, use Element.InsertTextBefore() or Element.InsertTextAfter(). To get an existing character run, use Element.GetElements(Element.Type.CharacterRun) or Element.Children and check which elements are of type Element.Type.CharacterRun.

The following sample demonstrates inserting a new character run into a document and retrieving the first character run from an existing document.

Examples

C#

```
//--- Insert a character run at the end of the document
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          CharacterRun run = doc.InsertTextAfter("Hello world!", true);

          //--- Get the first character run of an existing document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          CharacterRun firstCharRun =
               (CharacterRun)doc.GetElements(Element.Type.CharacterRun)[0];
```

vb.net

```
'--- Insert a character run at the end of the document
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim run As CharacterRun = doc.InsertTextAfter("Hello world!", True)

          '--- Get the first character run of an existing document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim firstCharRun As CharacterRun = _
               doc.GetElements(Elements.Type.CharacterRun)(0)
```

Properties

Name	Description
Deleted	Returns a `boolean` that represents if this character run was deleted with revision tracking on.
DeletedBy	Returns a `String` that represents the name of the author who deleted the text in this character run (if revision tracking was on when it was deleted -- Deleted returns true).
DeletedTime	Returns a `DateTime` object that represents the date and time the text in this character run was deleted (if revision tracking was on when it was deleted -- Deleted returns true).
Font	Sets or returns a font for the CharacterRun.
ModifiedBy	Returns a `String` that represents the name of the author who inserted the text in this character run (if revision tracking was on when it was modified).
ModifiedTime	Returns a `DateTime` object that represents the date and time the text in this character run was inserted (if revision tracking was on when it was modified).
New	Returns a `boolean` that represents if this character run was added while revision tracking was turned on.
PropertiesChangedBy	Returns a `String` that represents the name of the author who modified the properties of this character run.
PropertiesChangedTime	Returns a `DateTime` object that represents the date and time the properties of this character run were modified.
Style	Sets or Returns a NamedStyle object that represents the Style of this character run.

CharacterRun.Deleted

Description

Returns a `boolean` that represents if this character run was deleted with revision tracking on.

Signature: C#

```
public boolean Deleted{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Deleted() As Boolean
```

Examples

C#

```
bool deleted = oCharacterRun.Deleted;
```

vb.net

```
Dim deleted As Boolean = oCharacterRun.Deleted
```

CharacterRun.DeletedBy

Description

Returns a `String` that represents the name of the author who deleted the text in this character run (if revision tracking was on when it was deleted -- Deleted returns true).

Signature: C#

```
public System.String DeletedBy{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DeletedBy() As String
```

Examples

C#

```
string deletedBy = oCharacterRun.DeletedBy;
```

vb.net

```
Dim deletedBy As String = oCharacterRun.DeletedBy
```

CharacterRun.DeletedTime

Description

Returns a `DateTime` object that represents the date and time the text in this character run was deleted (if revision tracking was on when it was deleted -- Deleted returns true).

Signature: C#

```
public System.DateTime DeletedTime{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DeletedTime() As Date
```

Examples

C#

```
DateTime deletedTime = oCharacterRun.DeletedTime;
```

vb.net

```
Dim deletedTime As DateTime = oCharacterRun.DeletedTime
```

CharacterRun.Font

Description

Sets or returns a font for the CharacterRun.

Signature: C#

```
public Font Font{ get; set; }
```

Signature: VB.NET

```
Public Property Font() As Font
```

Examples

C#

```
//--- Get Font
          Font fnt = charRun.Font;

          //--- Set Font
          charRun.Font = fnt
```

vb.net

```
'--- Get Font
          Dim oFont As Font = charRun.Font

          '--- Set Font
          charRun.Font = fnt
```

CharacterRun.ModifiedBy

Description

Returns a `String` that represents the name of the author who inserted the text in this character run (if revision tracking was on when it was modified).

Signature: C#

```
public System.String ModifiedBy{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ModifiedBy() As String
```

Examples

C#

```
string modifiedBy = oCharacterRun.ModifiedBy;
```

vb.net

```
Dim modifiedBy As String = oCharacterRun.ModifiedBy
```

CharacterRun.ModifiedTime

Description

Returns a `DateTime` object that represents the date and time the text in this character run was inserted (if revision tracking was on when it was modified).

Signature: C#

```
public System.DateTime ModifiedTime{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ModifiedTime() As Date
```

Examples

C#

```
DateTime modifiedTime = oCharacterRun.ModifiedTime;
```

vb.net

```
Dim modifiedTime As DateTime = oCharacterRun.ModifiedTime
```

CharacterRun.New

Description

Returns a `boolean` that represents if this character run was added while revision tracking was turned on.

Signature: C#

```
public boolean New{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property [New]() As Boolean
```

Examples

C#

```
bool newRun = oCharacterRun.New;
```

vb.net

```
Dim newRun As Boolean = oCharacterRun.New
```

CharacterRun.PropertiesChangedBy

Description

Returns a `String` that represents the name of the author who modified the properties of this character run.

Signature: C#

```
public System.String PropertiesChangedBy{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PropertiesChangedBy() As String
```

Examples

C#

```
string propChangedBy = oCharacterRun.PropertiesChangedBy;
```

vb.net

```
Dim propChangedBy As String = oCharacterRun.PropertiesChangedBy
```

CharacterRun.PropertiesChangedTime

Description

Returns a `DateTime` object that represents the date and time the properties of this character run were modified.

Signature: C#

```
public System.DateTime PropertiesChangedTime{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PropertiesChangedTime() As Date
```

Examples

C#

```
DateTime propChangedTime = oCharacterRun.PropertiesChangedTime;
```

vb.net

```
Dim propChangedTime As DateTime = oCharacterRun.PropertiesChangedTime
```

CharacterRun.Style

Description

Sets or Returns a NamedStyle object that represents the Style of this character run.

Signature: C#

```
public NamedStyle Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As NamedStyle
```

CompatibilitySettings

Description

This class contains various settings so imported documents from other versions of Word and competing products will look the way they were intended to.

Signature: C#

```
public sealed class CompatibilitySettings
```

Signature: VB.NET

```
Public NotInheritable Class CompatibilitySettings
```

Remarks

You cannot create a CompatibilitySettings object using the `new` keyword. You must get an existing `CompatibilitySettings` object from a Document.

The following sample gets the CompatibilitySettings of a document so you can start working with these properties.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          CompatibilitySettings cs = doc.CompatibilitySettings;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim cs As CompatibilitySettings = doc.CompatibilitySettings
```

Properties

Name	Description
AdjustTableLineHeight	Sets or returns a `boolean` that represents whether or not Word will auto-adjust line height in Tables in this document.
AlignRowsIndependent	Sets or returns a `boolean` that represents whether or not Word will align table rows independently of one another in this document.
AllowLeading	Sets or returns a `boolean` that represents whether or not Word will add leading (extra space) between rows of text in this document.

AllowSnapInCells	Sets or returns a `boolean` that represents whether or not Word will snap text to grid while in a table with inline objects in this document.
AllowTableBreaks	Sets or returns a `boolean` that represents whether or not Word will allow tables to break across pages in this document.
ApplyBreakingRules	Sets or returns a `boolean` that represents whether or not Word will apply breaking rules in this document.
AutospaceLike95	Sets or returns a `boolean` that represents whether or not Word will auto space like Word 95 in this document.
BalanceColumns	Sets or returns a `boolean` that represents whether or not Word will balance columns for continuous section starts in this document.
BorderRulesLike6	Sets or returns a `boolean` that represents whether or not Word should use Word 6 border rules for this document.
ConvertMailMergeEsc	Sets or returns a `boolean` that represents whether or not Word will treat \" as "" in mail merge data sources in this document.
ExactOnTop	Sets or returns a `boolean` that represents whether or not Word will center "exact line height" lines in this document.
ExpandShiftReturn	Sets or returns a `boolean` that represents whether or not Word will expand character spaces on the line ending in SHIFT + RETURN in this document.
FootnotesLike97	Sets or returns a `boolean` that represents whether or not Word will layout footnotes like Word 97 in this document.
GrowTableAutoFit	Sets or returns a `boolean` that represents whether or not Word will allow autofit tables to extend into the margins when in print layout for this document.
JustifyLikeWP6	Sets or returns a `boolean` that represents whether or not Word will do full justification like WordPerfect 6 for this document.
LeaveBackslashAlone	Sets or returns a `boolean` that represents if Word will not convert '\' characters into Yen signs in this document.
LineBreaksLike97	Sets or returns a `boolean` that represents whether or not Word will use line breaking rules from Word 97 for East Asian text in this document.
LineWrapLike6	Sets or returns a `boolean` that represents whether or not Word will wrap lines like Word 6 for this document.
MapPrintColor	Sets or returns a `boolean` that represents whether or not Word will print colors in this document as black on non-color printers.
MirrorMargins	Sets or returns a `boolean` that represents whether or not Word will swap margins on left/right pages.
NoSpaceForRaisedLower	Sets or returns a `boolean` that represents if Word will not add extra space raised or lowered characters.
NoTabForIndent	Sets or returns a `boolean` that represents if Word will not add automatic tab stops for a hanging indent.
ShapesLike97	Sets or returns a `boolean` that represents whether or not Word will layout auto shapes like Word 97 for this document.
ShowBreaksInFrames	Sets or returns a `boolean` that represents whether or not Word will show hard page or column breaks in frames.
SnapPageBorder	Sets or returns a `boolean` that represents whether or not Word will snap table and page borders to the page border in this document.
SpaceLikeWP5	Sets or returns a `boolean` that represents whether or not Word will calculate the width of a space like WordPerfect 5 for this document.
SubBasedOnSize	Sets or returns a `boolean` that represents whether or not Word will replace fonts based on size in this document.
SuppressExtraAfter	Sets or returns a `boolean` that represents whether or not Word will supress extra line spacing at the bottom of the page for this document.
SuppressSpaceAfterPageBreak	Sets or returns a `boolean` that represents whether or not Word will suppress paragraph space before and space after options after a page break in this document.
SuppressSpacingLike5	Sets or returns a `boolean` that toggles the suppression of line spacing like Word 5 for Mac.
SuppressSpacingLikeWP	Sets or returns a `boolean` that toggles the suppression of line spacing like WordPerfect.

SupressTopSpacing	Sets or returns a `boolean` that represents whether or not Word will suppress line spacing at the top of the page in this document.
SwapBorders	Sets or returns a `boolean` that represents whether or not Word will swap left and right borders on odd facing pages in this document.
TableRowsApart	Sets or returns a `boolean` that represents whether or not Word will allow table rows to be layed out apart from one another in this document.
TruncateFontHeight	Sets or returns a `boolean` that represents whether or not Word will truncate font heights for this document.
UnderlineTrailingSpaces	Sets or returns a `boolean` that represents whether or not Word will underline trailing spaces for this document.
UseAsianBreakingRules	Sets or returns a `boolean` that represents whether or not Word will use Asian breaking rules for this document.
UseLargerSmallCaps	Sets or returns a `boolean` that represents whether or not Word will use larger small capitalization like Word 5 for Mac for this document.
UsePrinterMetrics	Sets or returns a `boolean` that represents whether or not Word will use printer metrics to lay out this document.
UseTableRawWidth	Sets or returns a `boolean` that represents whether or not Word will lay out tables using their raw width in this document.
UseTableStylesLike2002	Sets or returns a `boolean` that represents whether or not Word will use table styles like Word 2002. Word 2002 places the top border of a column under the heading row, rather than above it as Word 2003 does.
WrapTextWithPunct	Sets or returns a `boolean` that represents whether or not Word will wrap text with punctuation in this document.
WrapTrailingSpaces	Sets or returns a `boolean` that represents whether or not Word will wrap trailing spaces at the end of a line in this document.

CompatibilitySettings.AdjustTableLineHeight

Description

Sets or returns a `boolean` that represents whether or not Word will auto-adjust line height in Tables in this document.

Signature: C#

```
public boolean AdjustTableLineHeight{ get; set; }
```

Signature: VB.NET

```
Public Property AdjustTableLineHeight() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Adjust line height to grid height in the table

Examples

C#

```
//--- Return AdjustTableLineHeight
          bool adjustTableLineHeight = cs.AdjustTableLineHeight;

          //--- Set AdjustTableLineHeight
          cs.AdjustTableLineHeight = true;
```

vb.net

```
'--- Return AdjustTableLineHeight
          Dim adjustTableLineHeight As Boolean = oCompatibilitySettings.AdjustTableLineHeight

          '--- Set AdjustTableLineHeight
          oCompatibilitySettings.AdjustTableLineHeight = True
```

CompatibilitySettings.AlignRowsIndependent

Description

Sets or returns a `boolean` that represents whether or not Word will align table rows independently of one another in this document.

Signature: C#

```
public boolean AlignRowsIndependent{ get; set; }
```

Signature: VB.NET

```
Public Property AlignRowsIndependent() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Align table rows independently

Examples

C#

```
//--- Return AlignRowsIndependent
          bool alignRowsIndependent = cs.AlignRowsIndependent;

          //--- Set AlignRowsIndependent
          cs.AlignRowsIndependent = true;
```

vb.net

```
'--- Return AlignRowsIndependent
          Dim alignRowsIndependent As Boolean = cs.AlignRowsIndependent

          '--- Set AlignRowsIndependent
          cs.AlignRowsIndependent = True
```

CompatibilitySettings.AllowLeading

Description

Sets or returns a `boolean` that represents whether or not Word will add leading (extra space) between rows of text in this document.

Signature: C#

```
public boolean AllowLeading{ get; set; }
```

Signature: VB.NET

```
Public Property AllowLeading() As Boolean
```

Remarks

This is set to `true` by default.

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't balance columns for Continuous section starts

Examples

C#

```
//--- Return AllowLeading
          bool allowLeading = cs.AllowLeading;

          //--- Set AllowLeading
          cs.AllowLeading = true;
```

vb.net

```
'--- Return AllowLeading
          Dim allowLeading As Boolean = cs.AllowLeading

          '--- Set AllowLeading
          cs.AllowLeading = True
```

CompatibilitySettings.AllowSnapInCells

Description

Sets or returns a `boolean` that represents whether or not Word will snap text to grid while in a table with inline objects in this document.

Signature: C#

```
public boolean AllowSnapInCells{ get; set; }
```

Signature: VB.NET

```
Public Property AllowSnapInCells() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't snap text to grid inside table with Inline objects

Examples

C#

```
//--- Return AllowSnapInCells
          bool allowSnapInCells = cs.AllowSnapInCells;

          //--- Set AllowSnapInCells
          cs.AllowSnapInCells = true;
```

vb.net

```
'--- Return AllowSnapInCells
          Dim allowSnapInCells As Boolean = cs.AllowSnapInCells

          '--- Set AllowSnapInCells
          cs.AllowSnapInCells = True
```

CompatibilitySettings.AllowTableBreaks

Description

Sets or returns a `boolean` that represents whether or not Word will allow tables to break across pages in this document.

Signature: C#

```
public boolean AllowTableBreaks{ get; set; }
```

Signature: VB.NET

```
Public Property AllowTableBreaks() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't break wrapped tables across pages

Examples

C#

```
//--- Return AllowTableBreaks
          bool allowTableBreaks = cs.AllowTableBreaks;

          //--- Set AllowTableBreaks
          cs.AllowTableBreaks = true;
```

vb.net

```
'--- Return AllowTableBreaks
          Dim allowTableBreaks As Boolean = cs.AllowTableBreaks

          '--- Set AllowTableBreaks
          cs.AllowTableBreaks = True
```

CompatibilitySettings.ApplyBreakingRules

Description

Sets or returns a `boolean` that represents whether or not Word will apply breaking rules in this document.

Signature: C#

```
public boolean ApplyBreakingRules{ get; set; }
```

Signature: VB.NET

```
Public Property ApplyBreakingRules() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Use line breaking rules

Examples

C#

```
//--- Return ApplyBreakingRules
          bool applyBreakingRules = cs.ApplyBreakingRules;

          //--- Set ApplyBreakingRules
          cs.ApplyBreakingRules = true;
```

vb.net

```
'--- Return ApplyBreakingRules
          Dim applyBreakingRules As Boolean = cs.ApplyBreakingRules

          '--- Set ApplyBreakingRules
          cs.ApplyBreakingRules = True
```

CompatibilitySettings.AutospaceLike95

Description

Sets or returns a `boolean` that represents whether or not Word will auto space like Word 95 in this document.

Signature: C#

```
public boolean AutospaceLike95{ get; set; }
```

Signature: VB.NET

```
Public Property AutospaceLike95() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Auto space like Word 95

Examples

C#

```
//--- Return AutospaceLike95
          bool autospaceLike95 = cs.AutospaceLike95;

          //--- Set AutospaceLike95
          cs.AutospaceLike95 = true;
```

vb.net

```
'--- Return AutospaceLike95
          Dim autospaceLike95 As Boolean = cs.AutospaceLike95

          '--- Set AutospaceLike95
          cs.AutospaceLike95 = True
```

CompatibilitySettings.BalanceColumns

Description

Sets or returns a `boolean` that represents whether or not Word will balance columns for continuous section starts in this document.

Signature: C#

```
public boolean BalanceColumns{ get; set; }
```

Signature: VB.NET

```
Public Property BalanceColumns() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't balance columns for Continuous section starts

Examples

C#

```
//--- Return BalanceColumns
          bool balanceColumns = cs.BalanceColumns;

          //--- Set BalanceColumns
          cs.BalanceColumns = true;
```

vb.net

```
'--- Return BalanceColumns
          Dim balanceColumns As Boolean = cs.BalanceColumns

          '--- Set MapPrintColor
          cs.BalanceColumns = True
```

CompatibilitySettings.BorderRulesLike6

Description

Sets or returns a `boolean` that represents whether or not Word should use Word 6 border rules for this document.

Signature: C#

```
public boolean BorderRulesLike6{ get; set; }
```

Signature: VB.NET

```
Public Property BorderRulesLike6() As Boolean
```

Remarks

This is set to `true` by default.

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Use Word 6.x/95 Border rules

Examples

C#

```
//--- Return BorderRulesLike6
          bool borderRulesLike6 = cs.BorderRulesLike6;

          //--- Set BorderRulesLike6
          cs.BorderRulesLike6 = true;
```

vb.net

```
'--- Return BorderRulesLike6
          Dim borderRulesLike6 As Boolean = cs.BorderRulesLike6

          '--- Set BorderRulesLike6
          cs.BorderRulesLike6 = True
```

CompatibilitySettings.ConvertMailMergeEsc

Description

Sets or returns a `boolean` that represents whether or not Word will treat \" as "" in mail merge data sources in this document.

Signature: C#

```
public boolean ConvertMailMergeEsc{ get; set; }
```

Signature: VB.NET

```
Public Property ConvertMailMergeEsc() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Treat \" as "" in mail merge data sources

Examples

C#

```
//--- Return ConvertMailMergeEsc
          bool convertMailMergeEsc = cs.ConvertMailMergeEsc;

          //--- Set ConvertMailMergeEsc
          cs.ConvertMailMergeEsc = true;
```

vb.net

```
'--- Return ConvertMailMergeEsc
          Dim convertMailMergeEsc As Boolean = cs.ConvertMailMergeEsc

          '--- Set ConvertMailMergeEsc
          cs.ConvertMailMergeEsc = True
```

CompatibilitySettings.ExactOnTop

Description

Sets or returns a `boolean` that represents whether or not Word will center "exact line height" lines in this document.

Signature: C#

```
public boolean ExactOnTop{ get; set; }
```

Signature: VB.NET

```
Public Property ExactOnTop() As Boolean
```

Remarks

This is set to `false` by default.

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't center "exact line height" lines

Examples

C#

```
//--- Return ExactOnTop
          bool exactOnTop = cs.ExactOnTop;

          //--- Set ExactOnTop
          cs.ExactOnTop = true;
```

vb.net

```
'--- Return ExactOnTop
          Dim exactOnTop As Boolean = cs.ExactOnTop

          '--- Set ExactOnTop
          cs.ExactOnTop = True
```

CompatibilitySettings.ExpandShiftReturn

Description

Sets or returns a `boolean` that represents whether or not Word will expand character spaces on the line ending in SHIFT + RETURN in this document.

Signature: C#

```
public boolean ExpandShiftReturn{ get; set; }
```

Signature: VB.NET

```
Public Property ExpandShiftReturn() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't expand character spaces on the line ending Shift-Return

Examples

C#

```
//--- Return ExpandShiftReturn
          bool expandShiftReturn = cs.ExpandShiftReturn;

          //--- Set ExpandShiftReturn
          cs.ExpandShiftReturn = true;
```

vb.net

```
'--- Return ExpandShiftReturn
          Dim expandShiftReturn As Boolean = cs.ExpandShiftReturn

          '--- Set ExpandShiftReturn
          cs.ExpandShiftReturn = True
```

CompatibilitySettings.FootnotesLike97

Description

Sets or returns a `boolean` that represents whether or not Word will layout footnotes like Word 97 in this document.

Signature: C#

```
public boolean FootnotesLike97{ get; set; }
```

Signature: VB.NET

```
Public Property FootnotesLike97() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Lay out footnotes like Work 6.x/95/97

Examples

C#

```
//--- Return FootnotesLike97
          bool footnotesLike97 = cs.FootnotesLike97;

          //--- Set FootnotesLike97
          cs.FootnotesLike97 = true;
```

vb.net

```
'--- Return FootnotesLike97
          Dim footnotesLike97 As Boolean = cs.FootnotesLike97

          '--- Set FootnotesLike97
          cs.FootnotesLike97 = True
```

CompatibilitySettings.GrowTableAutoFit

Description

Sets or returns a `boolean` that represents whether or not Word will allow autofit tables to extend into the margins when in print layout for this document.

Signature: C#

```
public boolean GrowTableAutoFit{ get; set; }
```

Signature: VB.NET

```
Public Property GrowTableAutoFit() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Allow tables to extend into margins

Examples

C#

```
//--- Return GrowTableAutoFit
          bool growTableAutoFit = cs.GrowTableAutoFit;

          //--- Set GrowTableAutoFit
          cs.GrowTableAutoFit = true;
```

vb.net

```
'--- Return GrowTableAutoFit
          Dim growTableAutoFit As Boolean = cs.GrowTableAutoFit

          '--- Set GrowTableAutoFit
          cs.GrowTableAutoFit = True
```

CompatibilitySettings.JustifyLikeWP6

Description

Sets or returns a `boolean` that represents whether or not Word will do full justification like WordPerfect 6 for this document.

Signature: C#

```
public boolean JustifyLikeWP6{ get; set; }
```

Signature: VB.NET

```
Public Property JustifyLikeWP6() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Do full justification like WordPerfect 6.x for Windows

Examples

C#

```
//--- Return JustifyLikeWP6
          bool justifyLikeWP6 = cs.JustifyLikeWP6;

          //--- Set JustifyLikeWP6
          cs.JustifyLikeWP6 = true;
```

vb.net

```
'--- Return JustifyLikeWP6
          Dim justifyLikeWP6 As Boolean = cs.JustifyLikeWP6

          '--- Set JustifyLikeWP6
          cs.JustifyLikeWP6 = True
```

CompatibilitySettings.LeaveBackslashAlone

Description

Sets or returns a `boolean` that represents if Word will not convert '\' characters into Yen signs in this document.

Signature: C#

```
public boolean LeaveBackslashAlone{ get; set; }
```

Signature: VB.NET

```
Public Property LeaveBackslashAlone() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Convert backslash characters into yen signs

Examples

C#

```
//--- Return LeaveBackslashAlone
          bool leaveBackslashAlone = cs.LeaveBackslashAlone;

          //--- Set LeaveBackslashAlone
          cs.LeaveBackslashAlone = true;
```

vb.net

```
'--- Return LeaveBackslashAlone
          Dim leaveBackslashAlone As Boolean = cs.LeaveBackslashAlone

          '--- Set LeaveBackslashAlone
          cs.LeaveBackslashAlone = True
```

CompatibilitySettings.LineBreaksLike97

Description

Sets or returns a `boolean` that represents whether or not Word will use line breaking rules from Word 97 for East Asian text in this document.

Signature: C#

```
public boolean LineBreaksLike97{ get; set; }
```

Signature: VB.NET

```
Public Property LineBreaksLike97() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Use Word 97 line breaking rules for Asian text

Examples

C#

```
//--- Return LineBreaksLike97
          bool lineBreaksLike97 = cs.LineBreaksLike97;

          //--- Set LineBreaksLike97
          cs.LineBreaksLike97 = true;
```

vb.net

```
'--- Return LineBreaksLike97
          Dim lineBreaksLike97 As Boolean = cs.LineBreaksLike97

          '--- Set LineBreaksLike97
          cs.LineBreaksLike97 = True
```

CompatibilitySettings.LineWrapLike6

Description

Sets or returns a `boolean` that represents whether or not Word will wrap lines like Word 6 for this document.

Signature: C#

```
public boolean LineWrapLike6{ get; set; }
```

Signature: VB.NET

```
Public Property LineWrapLike6() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Lines wrap like Word 6.0

Examples

C#

```
//--- Return LineWrapLike6
          bool lineWrapLike6 = cs.LineWrapLike6;

          //--- Set LineWrapLike6
          cs.LineWrapLike6 = true;
```

vb.net

```
'--- Return LineWrapLike6
          Dim lineWrapLike6 As Boolean = cs.LineWrapLike6

          '--- Set LineWrapLike6
          cs.LineWrapLike6 = True
```

CompatibilitySettings.MapPrintColor

Description

Sets or returns a `boolean` that represents whether or not Word will print colors in this document as black on non-color printers.

Signature: C#

```
public boolean MapPrintColor{ get; set; }
```

Signature: VB.NET

```
Public Property MapPrintColor() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Print colors as black on noncolor printers

Examples

C#

```
//--- Return MapPrintColor
          bool mapPrintColor = cs.MapPrintColor;

          //--- Set MapPrintColor
          cs.MapPrintColor = true;
```

vb.net

```
'--- Return MapPrintColor
          Dim mapPrintColor As Boolean = cs.MapPrintColor

          '--- Set MapPrintColor
          cs.MapPrintColor = True
```

CompatibilitySettings.MirrorMargins

Description

Sets or returns a `boolean` that represents whether or not Word will swap margins on left/right pages.

Signature: C#

```
public boolean MirrorMargins{ get; set; }
```

Signature: VB.NET

```
Public Property MirrorMargins() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Swap left and right borders on odd facing pages

Examples

C#

```
//--- Return MirrorMargins
          bool mirrorMargins = cs.MirrorMargins;

          //--- Set MirrorMargins
          cs.MirrorMargins = true;
```

vb.net

```
'--- Return MirrorMargins
          Dim mirrorMargins As Boolean = cs.MirrorMargins

          '--- Set MirrorMargins
          cs.MirrorMargins = True
```

CompatibilitySettings.NoSpaceForRaisedLower

Description

Sets or returns a `boolean` that represents if Word will not add extra space raised or lowered characters.

Signature: C#

```
public boolean NoSpaceForRaisedLower{ get; set; }
```

Signature: VB.NET

```
Public Property NoSpaceForRaisedLower() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't add extra space for raised/lowered characters

Examples

C#

```
//--- Return NoSpaceForRaisedLower
          bool noSpaceForRaisedLower = cs.NoSpaceForRaisedLower;

          //--- Set NoTabForIndent
          cs.NoSpaceForRaisedLower = true;
```

vb.net

```
'--- Return NoSpaceForRaisedLower
          Dim noSpaceForRaisedLower As Boolean = cs.NoSpaceForRaisedLower

          '--- Set NoSpaceForRaisedLower
          cs.NoSpaceForRaisedLower = True
```

CompatibilitySettings.NoTabForIndent

Description

Sets or returns a `boolean` that represents if Word will not add automatic tab stops for a hanging indent.

Signature: C#

```
public boolean NoTabForIndent{ get; set; }
```

Signature: VB.NET

```
Public Property NoTabForIndent() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't add automatic tab stop for hanging indent

Examples

C#

```
//--- Return NoTabForIndent
          bool noTabForIndent = cs.NoTabForIndent;

          //--- Set NoTabForIndent
          cs.NoTabForIndent = true;
```

vb.net

```
'--- Return NoTabForIndent
          Dim noTabForIndent As Boolean = cs.NoTabForIndent

          '--- Set NoTabForIndent
          cs.NoTabForIndent = True
```

CompatibilitySettings.ShapesLike97

Description

Sets or returns a `boolean` that represents whether or not Word will layout auto shapes like Word 97 for this document.

Signature: C#

```
public boolean ShapesLike97{ get; set; }
```

Signature: VB.NET

```
Public Property ShapesLike97() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Lay out AutoShapes like Word 97

Examples

C#

```
//--- Return ShapesLike97
          bool shapesLike97 = cs.ShapesLike97;

          //--- Set ShapesLike97
          cs.ShapesLike97 = true;
```

vb.net

```
'--- Return ShapesLike97
          Dim shapesLike97 As Boolean = oCompatibilitySettings.ShapesLike97

          '--- Set ShapesLike97
          cs.ShapesLike97 = True
```

CompatibilitySettings.ShowBreaksInFrames

Description

Sets or returns a `boolean` that represents whether or not Word will show hard page or column breaks in frames.

Signature: C#

```
public boolean ShowBreaksInFrames{ get; set; }
```

Signature: VB.NET

```
Public Property ShowBreaksInFrames() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Show hard page or column breaks in frames

Examples

C#

```
//--- Return ShowBreaksInFrames
          bool showBreaksInFrames = cs.ShowBreaksInFrames;

          //--- Set ShowBreaksInFrames
          cs.ShowBreaksInFrames = true;
```

vb.net

```
'--- Return ShowBreaksInFrames
          Dim showBreaksInFrames As Boolean = cs.ShowBreaksInFrames

          '--- Set ShowBreaksInFrames
          cs.ShowBreaksInFrames = True
```

CompatibilitySettings.SnapPageBorder

Description

Sets or returns a `boolean` that represents whether or not Word will snap table and page borders to the page border in this document.

Signature: C#

```
public boolean SnapPageBorder{ get; set; }
```

Signature: VB.NET

```
Public Property SnapPageBorder() As Boolean
```

Examples

C#

```
//--- Return SnapPageBorder
          bool snapPageBorder = cs.SnapPageBorder;

          //--- Set SnapPageBorder
          cs.SnapPageBorder = true;
```

vb.net

```
'--- Return SnapPageBorder
          Dim snapPageBorder As Boolean = cs.SnapPageBorder

          '--- Set SnapPageBorder
          cs.SnapPageBorder = True
```

CompatibilitySettings.SpaceLikeWP5

Description

Sets or returns a `boolean` that represents whether or not Word will calculate the width of a space like WordPerfect 5 for this document.

Signature: C#

```
public boolean SpaceLikeWP5{ get; set; }
```

Signature: VB.NET

```
Public Property SpaceLikeWP5() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Set the width of a space like WordPerfect 5.x

Examples

C#

```
//--- Return SpaceLikeWP5
          bool spaceLikeWP5 = cs.SpaceLikeWP5;

          //--- Set SpaceLikeWP5
          cs.SpaceLikeWP5 = true;
```

vb.net

```
'--- Return SpaceLikeWP5
          Dim spaceLikeWP5 As Boolean = cs.SpaceLikeWP5

          '--- Set SpaceLikeWP5
          cs.SpaceLikeWP5 = True
```

CompatibilitySettings.SubBasedOnSize

Description

Sets or returns a `boolean` that represents whether or not Word will replace fonts based on size in this document.

Signature: C#

```
public boolean SubBasedOnSize{ get; set; }
```

Signature: VB.NET

```
Public Property SubBasedOnSize() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Substitute fonts based on font size

Examples

C#

```
//--- Return SubBasedOnSize
          bool subBasedOnSize = cs.SubBasedOnSize;

          //--- Set SubBasedOnSize
          cs.SubBasedOnSize = true;
```

vb.net

```
'--- Return SubBasedOnSize
          Dim subBasedOnSize As Boolean = cs.SubBasedOnSize

          '--- Set SubBasedOnSize
          cs.SubBasedOnSize = True
```

CompatibilitySettings.SuppressExtraAfter

Description

Sets or returns a `boolean` that represents whether or not Word will supress extra line spacing at the bottom of the page for this document.

Signature: C#

```
public boolean SuppressExtraAfter{ get; set; }
```

Signature: VB.NET

```
Public Property SuppressExtraAfter() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Suppress extra line spacing at bottom of page

Examples

C#

```
//--- Return SuppressExtraAfter
          bool suppressExtraAfter = cs.SuppressExtraAfter;

          //--- Set SuppressExtraAfter
          cs.SuppressExtraAfter = true;
```

vb.net

```
'--- Return SuppressExtraAfter
          Dim suppressExtraAfter As Boolean = cs.SuppressExtraAfter

          '--- Set SuppressExtraAfter
          cs.SuppressExtraAfter = True
```

CompatibilitySettings.SuppressSpaceAfterPageBreak

Description

Sets or returns a `boolean` that represents whether or not Word will suppress paragraph space before and space after options after a page break in this document.

Signature: C#

```
public boolean SuppressSpaceAfterPageBreak{ get; set; }
```

Signature: VB.NET

```
Public Property SuppressSpaceAfterPageBreak() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Suppress Space Before after a hard page or column break

Examples

C#

```
//--- Return SuppressSpaceAfterPageBreak
          bool suppressSpaceAfterPageBreak = cs.SuppressSpaceAfterPageBreak;

          //--- Set NoTabForIndent
          cs.SuppressSpaceAfterPageBreak = true;
```

vb.net

```
'--- Return SuppressSpaceAfterPageBreak
          Dim noSpaceForRaisedLower As Boolean = cs.SuppressSpaceAfterPageBreak

          '--- Set SuppressSpaceAfterPageBreak
          cs.SuppressSpaceAfterPageBreak = True
```

CompatibilitySettings.SuppressSpacingLike5

Description

Sets or returns a `boolean` that toggles the suppression of line spacing like Word 5 for Mac.

Signature: C#

```
public boolean SuppressSpacingLike5{ get; set; }
```

Signature: VB.NET

```
Public Property SuppressSpacingLike5() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Suppress extra line spacing at top of page like Word 5.x for the Mac

Examples

C#

```
//--- Return SuppressSpacingLike5
          bool suppressSpacingLike5 = cs.SuppressSpacingLike5;

          //--- Set SuppressSpacingLike5
          cs.SuppressSpacingLike5 = true;
```

vb.net

```
'--- Return SuppressSpacingLike5
          Dim suppressSpacingLike5 As Boolean = cs.SuppressSpacingLike5

          '--- Set SuppressSpacingLike5
          cs.SuppressSpacingLike5 = True
```

CompatibilitySettings.SuppressSpacingLikeWP

Description

Sets or returns a `boolean` that toggles the suppression of line spacing like WordPerfect.

Signature: C#

```
public boolean SuppressSpacingLikeWP{ get; set; }
```

Signature: VB.NET

```
Public Property SuppressSpacingLikeWP() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Suppress extra line spacing like WordPerfect 5.x

Examples

C#

```
//--- Return SuppressSpacingLikeWP
          bool suppressSpacingLikeWP = cs.SuppressSpacingLikeWP;

          //--- Set SuppressSpacingLikeWP
          cs.SuppressSpacingLikeWP = true;
```

vb.net

```
'--- Return SuppressSpacingLikeWP
          Dim suppressSpacingLikeWP As Boolean = cs.SuppressSpacingLikeWP

          '--- Set SuppressSpacingLikeWP
          cs.SuppressSpacingLikeWP = True
```

CompatibilitySettings.SupressTopSpacing

Description

Sets or returns a `boolean` that represents whether or not Word will suppress line spacing at the top of the page in this document.

Signature: C#

```
public boolean SupressTopSpacing{ get; set; }
```

Signature: VB.NET

```
Public Property SupressTopSpacing() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't balance columns for Continuous section starts

Examples

C#

```
//--- Return SupressTopSpacing
          bool supressTopSpacing = cs.SupressTopSpacing;

          //--- Set SupressTopSpacing
          cs.SupressTopSpacing = true;
```

vb.net

```
'--- Return SupressTopSpacing
          Dim supressTopSpacing As Boolean = cs.SupressTopSpacing

          '--- Set SupressTopSpacing
          cs.SupressTopSpacing = True
```

CompatibilitySettings.SwapBorders

Description

Sets or returns a `boolean` that represents whether or not Word will swap left and right borders on odd facing pages in this document.

Signature: C#

```
public boolean SwapBorders{ get; set; }
```

Signature: VB.NET

```
Public Property SwapBorders() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't balance columns for Continuous section starts

Examples

C#

```
//--- Return SwapBorders
          bool swapBorders = cs.SwapBorders;

          //--- Set SwapBorders
          cs.SwapBorders = true;
```

vb.net

```
'--- Return SwapBorders
          Dim swapBorders As Boolean = cs.SwapBorders

          '--- Set SupressTopSpacing
          cs.SwapBorders = True
```

CompatibilitySettings.TableRowsApart

Description

Sets or returns a `boolean` that represents whether or not Word will allow table rows to be layed out apart from one another in this document.

Signature: C#

```
public boolean TableRowsApart{ get; set; }
```

Signature: VB.NET

```
Public Property TableRowsApart() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Allow table rows to lay out apart

Examples

C#

```
//--- Return TableRowsApart
          bool tableRowsApart = cs.TableRowsApart;

          //--- Set TableRowsApart
          cs.TableRowsApart = true;
```

vb.net

```
'--- Return TableRowsApart
          Dim tableRowsApart As Boolean = cs.TableRowsApart

          '--- Set TableRowsApart
          cs.TableRowsApart = True
```

CompatibilitySettings.TruncateFontHeight

Description

Sets or returns a `boolean` that represents whether or not Word will truncate font heights for this document.

Signature: C#

```
public boolean TruncateFontHeight{ get; set; }
```

Signature: VB.NET

```
Public Property TruncateFontHeight() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Truncate font height

Examples

C#

```
//--- Return TruncateFontHeight
          bool truncateFontHeight = cs.TruncateFontHeight;

          //--- Set TruncateFontHeight
          cs.TruncateFontHeight = true;
```

vb.net

```
'--- Return TruncateFontHeight
          Dim truncateFontHeight As Boolean = cs.TruncateFontHeight

          '--- Set TruncateFontHeight
          cs.TruncateFontHeight = True
```

CompatibilitySettings.UnderlineTrailingSpaces

Description

Sets or returns a `boolean` that represents whether or not Word will underline trailing spaces for this document.

Signature: C#

```
public boolean UnderlineTrailingSpaces{ get; set; }
```

Signature: VB.NET

```
Public Property UnderlineTrailingSpaces() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't underline on trailing spaces

Examples

C#

```
//--- Return UnderlineTrailingSpaces
          bool underlineTrailingSpaces = cs.UnderlineTrailingSpaces;

          //--- Set UnderlineTrailingSpaces
          cs.UnderlineTrailingSpaces = true;
```

vb.net

```
'--- Return UnderlineTrailingSpaces
          Dim underlineTrailingSpaces As Boolean = cs.UnderlineTrailingSpaces

          '--- Set UnderlineTrailingSpaces
          cs.UnderlineTrailingSpaces = True
```

CompatibilitySettings.UseAsianBreakingRules

Description

Sets or returns a `boolean` that represents whether or not Word will use Asian breaking rules for this document.

Signature: C#

```
public boolean UseAsianBreakingRules{ get; set; }
```

Signature: VB.NET

```
Public Property UseAsianBreakingRules() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't use Asian rules for line breaks with character grid

Examples

C#

```
//--- Return UseAsianBreakingRules
          bool useAsianBreakingRules = cs.UseAsianBreakingRules;

          //--- Set UseAsianBreakingRules
          cs.UseAsianBreakingRules = true;
```

vb.net

```
'--- Return UseAsianBreakingRules
          Dim useAsianBreakingRules As Boolean = cs.UseAsianBreakingRules

          '--- Set UseAsianBreakingRules
          cs.UseAsianBreakingRules = True
```

CompatibilitySettings.UseLargerSmallCaps

Description

Sets or returns a `boolean` that represents whether or not Word will use larger small capitalization like Word 5 for Mac for this document.

Signature: C#

```
public boolean UseLargerSmallCaps{ get; set; }
```

Signature: VB.NET

```
Public Property UseLargerSmallCaps() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Use larger small caps like Word 5.x for the Macintosh

Examples

C#

```
//--- Return UseLargerSmallCaps
          bool useLargerSmallCaps = cs.UseLargerSmallCaps;

          //--- Set UseLargerSmallCaps
          cs.UseLargerSmallCaps = true;
```

vb.net

```
'--- Return UseLargerSmallCaps
          Dim useLargerSmallCaps As Boolean = cs.UseLargerSmallCaps

          '--- Set UseLargerSmallCaps
          cs.UseLargerSmallCaps = True
```

CompatibilitySettings.UsePrinterMetrics

Description

Sets or returns a `boolean` that represents whether or not Word will use printer metrics to lay out this document.

Signature: C#

```
public boolean UsePrinterMetrics{ get; set; }
```

Signature: VB.NET

```
Public Property UsePrinterMetrics() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Use printer metric to lay out document

Examples

C#

```
//--- Return UsePrinterMetrics
          bool usePrinterMetrics = cs.UsePrinterMetrics;

          //--- Set UsePrinterMetrics
          cs.UsePrinterMetrics = true;
```

vb.net

```
'--- Return UsePrinterMetrics
          Dim usePrinterMetrics As Boolean = oCompatibilitySettings.UsePrinterMetrics

          '--- Set UsePrinterMetrics
          cs.UsePrinterMetrics = True
```

CompatibilitySettings.UseTableRawWidth

Description

Sets or returns a `boolean` that represents whether or not Word will lay out tables using their raw width in this document.

Signature: C#

```
public boolean UseTableRawWidth{ get; set; }
```

Signature: VB.NET

```
Public Property UseTableRawWidth() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Lay out tables with raw width

Examples

C#

```
//--- Return UseTableRawWidth
          bool useTableRawWidth = cs.UseTableRawWidth;

          //--- Set UseTableRawWidth
          cs.UseTableRawWidth = true;
```

vb.net

```
'--- Return UseTableRawWidth
          Dim useTableRawWidth As Boolean = cs.UseTableRawWidth

          '--- Set UseTableRawWidth
          cs.UseTableRawWidth = True
```

CompatibilitySettings.UseTableStylesLike2002

Description

Sets or returns a `boolean` that represents whether or not Word will use table styles like Word 2002. Word 2002 places the top border of a column under the heading row, rather than above it as Word 2003 does.

Signature: C#

```
public boolean UseTableStylesLike2002{ get; set; }
```

Signature: VB.NET

```
Public Property UseTableStylesLike2002() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Use Word 2002 table style rules

Examples

C#

```
//--- Return UseTableStylesLike2002
          bool useTableStylesLike2002 = cs.UseTableStylesLike2002;

          //--- Set UseTableStylesLike2002
          cs.UseTableStylesLike2002 = true;
```

vb.net

```
'--- Return UseTableStylesLike2002
          Dim useTableStylesLike2002 As Boolean = cs.UseTableStylesLike2002

          '--- Set UseTableStylesLike2002
          cs.UseTableStylesLike2002 = True
```

CompatibilitySettings.WrapTextWithPunct

Description

Sets or returns a `boolean` that represents whether or not Word will wrap text with punctuation in this document.

Signature: C#

```
public boolean WrapTextWithPunct{ get; set; }
```

Signature: VB.NET

```
Public Property WrapTextWithPunct() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Don't allow hanging punctuation with character grid

Examples

C#

```
//--- Return WrapTextWithPunct
          bool wrapTextWithPunct = cs.WrapTextWithPunct;

          //--- Set WrapTextWithPunct
          cs.WrapTextWithPunct = true;
```

vb.net

```
'--- Return WrapTextWithPunct
          Dim wrapTextWithPunct As Boolean = cs.WrapTextWithPunct

          '--- Set WrapTextWithPunct
          cs.WrapTextWithPunct = True
```

CompatibilitySettings.WrapTrailingSpaces

Description

Sets or returns a `boolean` that represents whether or not Word will wrap trailing spaces at the end of a line in this document.

Signature: C#

```
public boolean WrapTrailingSpaces{ get; set; }
```

Signature: VB.NET

```
Public Property WrapTrailingSpaces() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Options... > Compatibility tab > Compatibility options for [filename].doc section > Options > Wrap trailing spaces to next line

Examples

C#

```
//--- Return WrapTrailingSpaces
          bool wrapTrailingSpaces = cs.WrapTrailingSpaces;

          //--- Set WrapTrailingSpaces
          cs.WrapTrailingSpaces = true;
```

vb.net

```
'--- Return WrapTrailingSpaces
          Dim wrapTrailingSpaces As Boolean = cs.WrapTrailingSpaces

          '--- Set WrapTrailingSpaces
          cs.WrapTrailingSpaces = True
```

DataImportProperties

Description

This class is used to control settings for importing data into a table or table cell.

Signature: C#

```
public sealed class DataImportProperties
```

Signature: VB.NET

```
Public NotInheritable Class DataImportProperties
```

Remarks

To create a DataImportProperties object, use Document.CreateDataImportProperties.

To actually import data, use Element.ImportData and pass in a `DataImportProperties` object along with the data to import.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          DataImportProperties importProps = doc.CreateDataImportProperties();

          //--- Set a few properties
          importProps.UseColumnNames = true;
          importProps.AutoFit = true;

          //--- Import data into a new table and apply the DataImportProperties
          Table dataImportTable = oTable.ImportData(oDataTable, importProps);
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim importProps As DataImportProperties = doc.CreateDataImportProperties()

          '--- Set a few properties
          importProps.UseColumnNames = True
          importProps.AutoFit = True

          '--- Import data into a new table and apply the DataImportProperties
          Table dataImportTable = oTable.ImportData(oDataTable, importProps)
```

Properties

Name	Description
AutoFit	Sets or returns a boolean representing whether or not the table that is automatically created for the imported data will automatically be sized to fit the data being imported.
AutoFormatter	Sets or returns an AutoFormatter that specifies how the imported data should be formatted in the table.
ColumnIndexFilter	Sets or returns an array of column indexes specifying the columns to import.
ColumnNameFilter	Sets or returns an array of columns names specifying the columns to import. If the length of the specified array is greater than MaxColumns, or exceeds the size of the target area, the columns will be added in the order in which they appear in the array.
MaxColumns	Sets or returns an `int` that represents the maximum number of columns to be imported. If more columns than this are in the data source, the extra ones will be ignored.
MaxRows	Returns or set an `int` that represents the maximum number of rows to be imported. If more rows than this are in the data source, the extra ones will be ignored.
UseColumnNames	Sets or returns a `boolean` that represents whether to use the column names from the data source when importing the data. This property is set to true by default.

DataImportProperties.AutoFit

Description

Sets or returns a boolean representing whether or not the table that is automatically created for the imported data will automatically be sized to fit

the data being imported.

Signature: C#

```
public boolean AutoFit{ get; set; }
```

Signature: VB.NET

```
Public Property AutoFit() As Boolean
```

Remarks

This is set to `false` by default.

Examples

C#

```
//---- Return AutoFit
          boolean fit = importProps.AutoFit;

          //--- Set AutoFit
          importProps.AutoFit = true;
```

vb.net

```
'--- Return AutoFit
          Dim fit As Boolean = importProps.AutoFit

          '--- Set AutoFit
          importProps.AutoFit = True
```

DataImportProperties.AutoFormatter

Description

Sets or returns an AutoFormatter that specifies how the imported data should be formatted in the table.

Signature: C#

```
public AutoFormatter AutoFormatter{ get; set; }
```

Signature: VB.NET

```
Public Property AutoFormatter() As AutoFormatter
```

DataImportProperties.ColumnIndexFilter

Description

Sets or returns an array of column indexes specifying the columns to import.

Signature: C#

```
public int[] ColumnIndexFilter{ get; set; }
```

Signature: VB.NET

```
Public Property ColumnIndexFilter() As Integer()
```

Remarks

If the length of the specified array is greater than MaxColumns, or exceeds the size of the target area, the columns will be added in the order in which they appear in the array.

Examples

C#

```
//--- Return ColumnIndexFilter
          int[] colIndexes = importProps.ColumnIndexFilter;

          //--- Set ColumnIndexFilter
          importProps.ColumnIndexFilter = new int[]{0,2,3,6};
```

vb.net

```
'--- Return ColumnIndexFilter
          Dim colIndexes() As Integer = importProps.ColumnIndexFilter

          '--- Set ColumnIndexFilter
          importProps.ColumnIndexFilter = New Integer(){0, 2, 3, 6}
```

DataImportProperties.ColumnNameFilter

Description

Sets or returns an array of columns names specifying the columns to import. If the length of the specified array is greater than MaxColumns, or exceeds the size of the target area, the columns will be added in the order in which they appear in the array.

Signature: C#

```
public System.String[] ColumnNameFilter{ get; set; }
```

Signature: VB.NET

```
Public Property ColumnNameFilter() As String()
```

Examples

C#

```
//--- Return ColumnNameFilter
          String[] colNames = importProps.ColumnNameFilter;

          //--- Set ColumnNameFilter
          importProps.ColumnNameFilter = new String[]{"Name", "Address", "Zip"};
```

vb.net

```
'--- Return ColumnNameFilter
          Dim colNames As String() = importProps.ColumnNameFilter

          '--- Set ColumnNameFilter
          importProps.ColumnNameFilter = New String() {"Name", "Address", "Zip"}
```

DataImportProperties.MaxColumns

Description

Sets or returns an `int` that represents the maximum number of columns to be imported. If more columns than this are in the data source, the extra ones will be ignored.

Signature: C#

```
public int MaxColumns{ get; set; }
```

Signature: VB.NET

```
Public Property MaxColumns() As Integer
```

Remarks

The default value is 0, which represents unlimited columns.

Examples

C#

```
//--- Return MaxColumns
          int cols = importProps.MaxColumns;

          //--- Set MaxRows
          importProps.MaxColumns = 10;
```

vb.net

```
'--- Return MaxColumns
          Dim cols As Integer = importProps.MaxColumns

          '--- Set MaxRows
          importProps.MaxColumns = 10
```

DataImportProperties.MaxRows

Description

Returns or set an `int` that represents the maximum number of rows to be imported. If more rows than this are in the data source, the extra ones will be ignored.

Signature: C#

```
public int MaxRows{ get; set; }
```

Signature: VB.NET

```
Public Property MaxRows() As Integer
```

Examples

C#

```
//--- Return MaxRows
          int rows = importProps.MaxRows;

          //--- Set MaxRows
          importProps.MaxRows = 50;
```

vb.net

```
'--- Return MaxRows
          Dim rows As Integer = importProps.MaxRows

          '--- Set MaxRows
          importProps.MaxRows = 50
```

DataImportProperties.UseColumnNames

Description

Sets or returns a `boolean` that represents whether to use the column names from the data source when importing the data. This property is set to true by default.

Signature: C#

```
public boolean UseColumnNames{ get; set; }
```

Signature: VB.NET

```
Public Property UseColumnNames() As Boolean
```

Examples

C#

```
//--- Return UseColumnNames
          bool useColNames = importProps.UseColumnNames;

          //--- Set UseColumnNames
          importProps.UseColumnNames = false;
```

vb.net

```
'--- Return UseColumnNames
          Boolean useColNames = importProps.UseColumnNames

          '--- Set UseColumnNames
          importProps.UseColumnNames = False
```

Document

Description

Represents a Word document.

Signature: C#

```
public sealed class Document : Element
```

Signature: VB.NET

```
Public NotInheritable Class Document
  Inherits Element
```

Remarks

To create a new document, use WordApplication.Create(). To open an existing document, use WordApplication.Open().

Note: Document inherits from Element and inherits a lot basic functionality from that class. Be sure to see the `Element` API docs when looking for specific functionality.

Examples

C#

```
//--- Create a new Word document
          WordApplication app = new WordApplication();
          Document doc = app.Create();

          //--- Open an existing Word document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
```

vb.net

```
'--- Create a new Word document
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()

          '--- Open an existing Word document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
```

Properties

Name	Description
AutoformatType	Sets or returns a Document.Autoformat object representing autoformat type for this document. Possible values: Normal, Letter, Email.
CompatibilitySettings	Returns a CompatibilitySettings object that can be used to get and set various settings that help Word display documents that were imported from earlier versions and competing products.
ChangeTracking	Returns a WordChangeTracking object that can be used to get and set the change tracking settings for this Document.
DateCreated	Returns a `DateTime` object representing the date this document was created.
DatePrinted	Returns a `DateTime` object representing the date this document was last printed using Word.
DateRevised	Returns a `DateTime` object representing the date this document was last revised.
DefaultTabWidth	Sets or returns an `int` representing the default tab width for this Document. The value that is returned is in twips. One twip = (1/20 pt) or (1/1440 in).
DocumentProperties	Returns a Styles object that represents the properties of this document.
DocumentProtection	Returns a DocumentProtection object that can be used to get and set the protection settings that Word will apply to this document. (Only applicable to Word 2000 or above documents.)
GutterPosition	Sets or returns a Document.Gutter object representing the gutter position in this document when it is opened in Word. If you have set up your document with facing pages or two pages per sheet (by using Mirror margins, Book fold, or 2 pages per sheet), then this option is not available. Possible values: Side, Top.
HyphenSettings	Returns a WordHyphenSettings object that can be used to get and set the hyphenation settings for this Document .
MergeStylesFromTemplate	Sets or returns a `boolean` representing if Word will update the styles in the current document to match the styles in the attached Word template (i.e. Normal.dot) each time you open the document in Word. This should be set to `true` when you want to ensure that your document contains up-to-date style formatting.
NumCharacters	Returns an `int` representing the number of characters in this document.
NumPagesOnLastOpen	Returns an `int` representing the number of pages this document had when it was last opened in Word.

NumParagraphs	Returns an `int` representing the number of paragraphs in this document.
NumWords	Returns an `int` representing the number of words in this document.
PrintDataInFormFields	Sets or returns a `boolean` representing if Word will only print the data inside a form field in this document. This is typically used to print the data entered into an online form without printing the online form.
PrintFacingPages	Sets or returns a `boolean` representing if the headers and footers should be different for odd and even pages, such as when the pages in this document will be mirrored. When set to `true`, it creates one header or footer for even-numbered pages and a different header or footer for odd-numbered pages.
SaveDataInFormField	Sets or returns a `boolean` representing if Word will save only the data entered into a form field (instead of the field itself) for this document. When this is set to `true`, Word saves the data entered into an online form as a single, tab-delimited record so you can use it in a database. Word saves the file in Text Only file format.
Sections	Returns an array of Section objects that represents all the sections in this Document.
Styles	Returns a Styles object that represents the Styles embedded in this document.
UseWidowControl	Sets or returns a `boolean` representing if widow control is in effect for this Document.
ViewSettings	Returns a WordViewSettings object that can be used to get and set various settings that pertain to how Word will display this document.
Watermark	Returns a Watermark object representing an image or text that appears in the background of a printed document.

Methods

Name	Description
Append(Document)	Appends an existing Document to the end of the current Document, separated by a next-page section break
CreateDataImportProperties()	Returns a DataImportProperties object, which you can then use to set options for importing data into a table in a Word document.
CreateFont()	Returns a Font object based on the font used by the Normal Style. You can then change the properties of this font and use it elsewhere.
CreateParagraphFormatting()	Returns a ParagraphFormatting object based on the paragraph formatting used by the BodyText style. You can then change the properties of this font and use it elsewhere.
CreateTableFormatting()	Returns a TableFormatting object that represents the table formatting used by the NormalTable style.
GetBookmark(String)	Returns a Bookmark object that represents the bookmark in this document that has the specified name.
GetVariable(String)	Returns the document variable specified by name.
SetVariable(String, String)	Sets the value of the variable with the specified name. If the variable does not exist, `setVariable` creates a new variable. If the variable does exist, `setVariable` overwrites the existing variable.

Nested Classes

Name	Description
Autoformat	A Document.Autoformat value specifies the type of AutoFormat settings to apply to a document. To select an AutoFormat type in Microsoft Word:
Gutter	A Document.Gutter value specifies the position of a document's gutter margin. Gutter margins are used in documents that will be bound. A gutter margin is an extra margin on the top or side of the page that ensures that the text is not obscured when the document is bound.

Document.Append(Document)

Introduced in build 4.1.0.1494

Description

Appends an existing Document to the end of the current Document, separated by a next-page section break

Signature: C#

```
public void Append(SoftArtisans.OfficeWriter.WordWriter.Document other)
```

Signature: VB.NET

```
Public Sub Append(ByVal other As SoftArtisans.OfficeWriter.WordWriter.Document)
```

Parameters

other

Document object being appended to the current document.

Remarks

This method works by adding all of the Sections of the appended document to the current document; if a the current document has 3 sections and the appended document has 4 sections, the resulting document will have 7 sections. Because sections are preserved, headers, footers, and all other section formatting from both documents are also preserved.

Examples

C#

```
//--- Open first document
        Document doc1 = wa.Open(Page.MapPath("documents/first document.doc"));

        //--- Open second document
        Document doc2 = wa.Open(Page.MapPath("documents/second document.doc"));

        //--- Append second document to the first
        doc1.Append(doc2);

        //--- Save the resulting document and stream to customer
        wa.Save(doc1, Page.Response, "output.doc", false);
```

vb.net

```
'--- Open first document
        Dim doc1 As Document = wa.Open(Page.MapPath("documents/first document.doc"))

        '--- Open second document
        Dim doc2 As Document = wa.Open(Page.MapPath("documents/second document.doc"))

        '--- Append second document to the first
        doc1.Append(doc2)

        '--- Save the resulting document and stream to customer
        wa.Save(doc1, Page.Response, "output.doc", false)
```

Document.Autoformat

Description

A Document.Autoformat value specifies the type of AutoFormat settings to apply to a document. To select an AutoFormat type in Microsoft Word:

Signature: C#

```
public sealed class Autoformat : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Autoformat
  Inherits TypesafeEnum
```

Values

Name	Description
Email	The document is an email message
Letter	The document is a letter
Normal	General document

Document.AutoformatType

Description

Sets or returns a Document.Autoformat object representing autoformat type for this document. Possible values: Normal, Letter, Email.

Signature: C#

```
public Autoformat AutoformatType{ get; set; }
```

Signature: VB.NET

```
Public Property AutoformatType() As Autoformat
```

Remarks

The default value for a new Document is Autoformat.Normal.

MS Word equivalent: Format menu > Autoformat... > Please select a document type to help improve the formatting process

Examples

C#

```
//--- Return AutoformatType
          Document.Autoformat oAutoformatType = oDocument.AutoformatType;

          //--- Set AutoformatType
          oDocument.AutoformatType = Document.Autoformat.Letter;
```

vb.net

```
'--- Return AutoformatType
          Dim oAutoformatType As Document.Autoformat = oDocument.AutoformatType

          '--- Set AutoformatType
          oDocument.AutoformatType = Document.Autoformat.Letter
```

Document.ChangeTracking

Description

Returns a WordChangeTracking object that can be used to get and set the change tracking settings for this Document.

Signature: C#

```
public WordChangeTracking ChangeTracking{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ChangeTracking() As WordChangeTracking
```

Examples

C#

```
WordChangeTracking oWordChangeTracking = oDocument.ChangeTracking;
```

vb.net

```
Dim oWordChangeTracking As WordChangeTracking = oDocument.ChangeTracking
```

Document.CompatibilitySettings

Description

Returns a CompatibilitySettings object that can be used to get and set various settings that help Word display documents that were imported from earlier versions and competing products.

Signature: C#

```
public CompatibilitySettings CompatibilitySettings{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property CompatibilitySettings() As CompatibilitySettings
```

Examples

C#

```
CompatibilitySettings oCompatibilitySettings = oDocument.CompatibilitySettings;
```

vb.net

```
Dim oCompatibilitySettings As CompatibilitySettings = oDocument.CompatibilitySettings
```

Document.CreateDataImportProperties()

Description

Returns a DataImportProperties object, which you can then use to set options for importing data into a table in a Word document.

Signature: C#

```
public DataImportProperties CreateDataImportProperties()
```

Signature: VB.NET

```
Public Function CreateDataImportProperties() As DataImportProperties
```

Returns

A `DataImportProperties` object, which you can then use to set options for importing data into a table in a Word document.

Examples

C#

```
DataImportProperties oDataImportProperties = oDocument.CreateDataImportProperties();
```

vb.net

```
Dim oDataImportProperties As DataImportProperties = oDocument.CreateDataImportProperties()
```

Document.CreateFont()

Description

Returns a Font object based on the font used by the Normal Style. You can then change the properties of this font and use it elsewhere.

Signature: C#

```
public Font CreateFont()
```

Signature: VB.NET

```
Public Function CreateFont() As Font
```

Returns

A `Font` object based on the font used by the Normal Style.

Examples

C#

```
Font oFont = oDocument.CreateFont();
```

vb.net

```
Dim oFont As Font = oDocument.CreateFont()
```

Document.CreateParagraphFormatting()

Description

Returns a ParagraphFormatting object based on the paragraph formatting used by the BodyText style. You can then change the properties of this font and use it elsewhere.

Signature: C#

```
public ParagraphFormatting CreateParagraphFormatting()
```

Signature: VB.NET

```
Public Function CreateParagraphFormatting() As ParagraphFormatting
```

Returns

A `ParagraphFormatting` object based on the paragraph formatting used by the BodyText style.

Examples

C#

```
ParagraphFormatting oParagraphFormatting = oDocument.CreateParagraphFormatting();
```

vb.net

```
Dim oParagraphFormatting As ParagraphFormatting = oDocument.CreateParagraphFormatting()
```

Document.CreateTableFormatting()

Description

Returns a TableFormatting object that represents the table formatting used by the NormalTable style.

Signature: C#

```
public TableFormatting CreateTableFormatting()
```

Signature: VB.NET

```
Public Function CreateTableFormatting() As TableFormatting
```

Returns

A `TableFormatting` object that represents the table formatting used by the NormalTable style.

Examples

C#

```
TableFormatting oTableFormatting = oDocument.CreateTableFormatting();
```

vb.net

```
Dim oTableFormatting As TableFormatting = oDocument.CreateTableFormatting()
```

Document.DateCreated

Description

Returns a `DateTime` object representing the date this document was created.

Signature: C#

```
public System.DateTime DateCreated{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DateCreated() As Date
```

Remarks

MS Word equivalent: File menu > Properties > Statistics tab > Created:

Examples

C#

```
DateTime oDateTime = oDocument.DateCreated;
```

vb.net

```
Dim oDateTime As DateTime = oDocument.DateCreated
```

Document.DatePrinted

Description

Returns a `DateTime` object representing the date this document was last printed using Word.

Signature: C#

```
public System.DateTime DatePrinted{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DatePrinted() As Date
```

Remarks

MS Word equivalent: File menu > Properties > Statistics tab > Printed:

Examples

C#

```
DateTime oDateTime = oDocument.DatePrinted;
```

vb.net

```
Dim oDateTime As DateTime = oDocument.DatePrinted
```

Document.DateRevised

Description

Returns a `DateTime` object representing the date this document was last revised.

Signature: C#

```
public System.DateTime DateRevised{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DateRevised() As Date
```

Remarks

MS Word equivalent: File menu > Properties > Statistics tab > Modified:

Examples

C#

```
DateTime oDateTime = oDocument.DateRevised;
```

vb.net

```
Dim oDateTime As DateTime = oDocument.DateRevised
```

Document.DefaultTabWidth

Description

Sets or returns an `int` representing the default tab width for this Document. The value that is returned is in twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int DefaultTabWidth{ get; set; }
```

Signature: VB.NET

```
Public Property DefaultTabWidth() As Integer
```

Remarks

MS Word equivalent: Format menu > Tabs... > Default tab stops

Examples

C#

```
//--- Return DefaultTabWidth
          int defaultTabWidth = oDocument.DefaultTabWidth;

          //--- Set DefaultTabWidth to 1 inch
          oDocument.DefaultTabWidth = 1440;
```

vb.net

```
'--- Return DefaultTabWidth
          Dim defaultTabWidth As Integer = oDocument.DefaultTabWidth

          '--- Set DefaultTabWidth to 1 inch
          oDocument.DefaultTabWidth = 1440
```

Document.DocumentProperties

Description

Returns a Styles object that represents the properties of this document.

Signature: C#

```
public DocumentProperties DocumentProperties{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DocumentProperties() As DocumentProperties
```

Remarks

MS Word equivalent: File menu > Properties

Examples

C#

```
DocumentProperties oDocumentProperties = oDocument.DocumentProperties;
```

vb.net

```
Dim oDocumentProperties As DocumentProperties = oDocument.DocumentProperties
```

Document.DocumentProtection

Description

Returns a DocumentProtection object that can be used to get and set the protection settings that Word will apply to this document. (Only applicable to Word 2000 or above documents.)

Signature: C#

```
public DocumentProtection DocumentProtection{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DocumentProtection() As DocumentProtection
```

Remarks

MS Word equivalent: Tools menu > Protect document... (plus some additional settings)

Examples

C#

```
DocumentProtection oDocumentProtection = oDocument.DocumentProtection;
```

vb.net

```
Dim oDocumentProtection As DocumentProtection = oDocument.DocumentProtection
```

Document.GetBookmark(String)

Description

Returns a Bookmark object that represents the bookmark in this document that has the specified name.

Signature: C#

```
public Bookmark GetBookmark(System.String name)
```

Signature: VB.NET

```
Public Function GetBookmark(ByVal name As String) As Bookmark
```

Parameters

name

The name of the bookmark.

Returns

A `Bookmark` object representing the bookmark specified.

Examples

C#

```
Bookmark oBookmark = oDocument.GetBookmark("MyBookmark");
```

vb.net

```
Dim oBookmark As Bookmark = oDocument.GetBookmark("MyBookmark")
```

Document.GetVariable(String)

Description

Returns the document variable specified by name.

Signature: C#

```
public System.String GetVariable(System.String name)
```

Signature: VB.NET

```
Public Function GetVariable(ByVal name As String) As String
```

Parameters

name

The name of the variable.

Returns

The document variable specified by name. If the variable does not exist, GetVariable returns null.

Examples

C#

```
string docVariable = doc.GetVariable("var");
```

vb.net

```
Dim docVariable As String = doc.GetVariable("var")
```

Document.Gutter

Description

A Document.Gutter value specifies the position of a document's gutter margin. Gutter margins are used in documents that will be bound. A gutter margin is an extra margin on the top or side of the page that ensures that the text is not obscured when the document is bound.

Signature: C#

```
public sealed class Gutter : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Gutter
  Inherits TypesafeEnum
```

Values

Name	Description
Side	The gutter margin will be on the left side of odd numbered pages and the right side of even numbered pages.
Top	The gutter margin will be at the top of the document.

Document.GutterPosition

Description

Sets or returns a Document.Gutter object representing the gutter position in this document when it is opened in Word. If you have set up your document with facing pages or two pages per sheet (by using Mirror margins, Book fold, or 2 pages per sheet), then this option is not available. Possible values: Side, Top.

Signature: C#

```
public Gutter GutterPosition{ get; set; }
```

Signature: VB.NET

```
Public Property GutterPosition() As Gutter
```

Remarks

The default value for a new Document is Document.Gutter.Side.

MS Word equivalent: File menu > Page Setup... > Margins tab > Margins section > Gutter position:

Examples

C#

```
//--- Return GutterPosition
          Document.Gutter oGutterPosition = oDocument.GutterPosition;

          //--- Set GutterPosition
          oDocument.GutterPosition = Document.Gutter.Top;
```

vb.net

```
'--- Return GutterPosition
          Dim oGutterPosition As Document.Gutter = oDocument.GutterPosition

          '--- Set GutterPosition
          oDocument.GutterPosition = Gutter.Top
```

Document.HyphenSettings

Description

Returns a WordHyphenSettings object that can be used to get and set the hyphenation settings for this Document.

Signature: C#

```
public WordHyphenSettings HyphenSettings{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property HyphenSettings() As WordHyphenSettings
```

Examples

C#

```
WordHyphenSettings oWordHyphenSettings = oDocument.HyphenSettings;
```

vb.net

```
Dim oWordHyphenSettings As WordHyphenSettings = oDocument.HyphenSettings
```

Document.MergeStylesFromTemplate

Description

Sets or returns a `boolean` representing if Word will update the styles in the current document to match the styles in the attached Word template (i.e. Normal.dot) each time you open the document in Word. This should be set to `true` when you want to ensure that your document contains up-to-date style formatting.

Signature: C#

```
public boolean MergeStylesFromTemplate{ get; set; }
```

Signature: VB.NET

```
Public Property MergeStylesFromTemplate() As Boolean
```

Remarks

The default value for a new Document is `false`.

MS Word equivalent: Tools menu > Templates and Add-ins... > Document template section > Automatically update document styles

Examples

C#

```
//--- Return MergeStylesFromTemplate
          bool mergeStyles = oDocument.MergeStylesFromTemplate;

          //--- Set MergeStylesFromTemplate
          oDocument.MergeStylesFromTemplate = true;
```

vb.net

```
'--- Return MergeStylesFromTemplate
          Dim mergeStyles As Boolean = oDocument.MergeStylesFromTemplate

          '--- Set MergeStylesFromTemplate
          oDocument.MergeStylesFromTemplate = True
```

Document.NumCharacters

Description

Returns an `int` representing the number of characters in this document.

Signature: C#

```
public int NumCharacters{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NumCharacters() As Integer
```

Remarks

MS Word equivalent: File menu > Properties > Statistics tab > Statistics listbox > Characters:

Examples

C#

```
int numCharacters = oDocument.NumCharacters;
```

vb.net

```
Dim numCharacters As Integer = oDocument.NumCharacters
```

Document.NumPagesOnLastOpen

Description

Returns an `int` representing the number of pages this document had when it was last opened in Word.

Signature: C#

```
public int NumPagesOnLastOpen{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NumPagesOnLastOpen() As Integer
```

Remarks

MS Word equivalent: File menu > Properties > Statistics tab > Statistics listbox > Pages:

Examples

C#

```
int numPages = oDocument.NumPagesOnLastOpen;
```

vb.net

```
Dim numPages As Integer = oDocument.NumPagesOnLastOpen
```

Document.NumParagraphs

Description

Returns an `int` representing the number of paragraphs in this document.

Signature: C#

```
public int NumParagraphs{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NumParagraphs() As Integer
```

Remarks

MS Word equivalent: File menu > Properties > Statistics tab > Statistics listbox > Paragraphs:

Examples

C#

```
int numParagraphs = oDocument.NumParagraphs;
```

vb.net

```
Dim numParagraphs As Integer = oDocument.NumParagraphs
```

Document.NumWords

Description

Returns an `int` representing the number of words in this document.

Signature: C#

```
public int NumWords{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NumWords() As Integer
```

Remarks

MS Word equivalent: File menu > Properties > Statistics tab > Statistics listbox > Words:

Examples

C#

```
int numWords = oDocument.NumWords;
```

vb.net

```
Dim numWords As Integer = oDocument.NumWords
```

Document.PrintDataInFormFields

Description

Sets or returns a `boolean` representing if Word will only print the data inside a form field in this document. This is typically used to print the data entered into an online form without printing the online form.

Signature: C#

```
public boolean PrintDataInFormFields{ get; set; }
```

Signature: VB.NET

```
Public Property PrintDataInFormFields() As Boolean
```

Remarks

The default value for a new Document is `false`.

MS Word equivalent: File menu > Print... > Options... button > Options for current document only section > Print data only for forms
Or
File menu > Page Setup > Paper tab > Print options... button > Options for current document only section > Print data only for forms

Examples

C#

```
//--- Return PrintDataInFormFields
          bool printData = oDocument.PrintDataInFormFields;

          //--- Set PrintDataInFormFields
          oDocument.PrintDataInFormFields = true;
```

vb.net

```
'--- Return PrintDataInFormFields
          Dim printData As Boolean = oDocument.PrintDataInFormFields

          '--- Set PrintDataInFormFields
          oDocument.PrintDataInFormFields = True
```

Document.PrintFacingPages

Description

Sets or returns a `boolean` representing if the headers and footers should be different for odd and even pages, such as when the pages in this document will be mirrored. When set to `true`, it creates one header or footer for even-numbered pages and a different header or footer for odd-numbered pages.

Signature: C#

```
public boolean PrintFacingPages{ get; set; }
```

Signature: VB.NET

```
Public Property PrintFacingPages() As Boolean
```

Remarks

MS Word equivalent: File menu > Page Setup... > Layout tab > Headers and footers section > Different odd and even

Examples

C#

```
//--- Return PrintFacingPages
          bool printFacingPages = oDocument.PrintFacingPages;

          //--- Set PrintFacingPages
          oDocument.PrintFacingPages = true;
```

vb.net

```
'--- Return PrintFacingPages
          Dim printFacingPages As Boolean = oDocument.PrintFacingPages

          '--- Set PrintFacingPages
          oDocument.PrintFacingPages = True
```

Document.SaveDataInFormField

Description

Sets or returns a `boolean` representing if Word will save only the data entered into a form field (instead of the field itself) for this document. When this is set to `true`, Word saves the data entered into an online form as a single, tab-delimited record so you can use it in a database. Word saves the file in Text Only file format.

Signature: C#

```
public boolean SaveDataInFormField{ get; set; }
```

Signature: VB.NET

```
Public Property SaveDataInFormField() As Boolean
```

Remarks

The default value for a new Document is `false`.

MS Word equivalent: Tools menu > Options... > Save tab > Save options section > Save data only for forms

Examples

C#

```
//--- Return SaveDataInFormField
          bool vSaveData = oDocument.SaveDataInFormField;

          //--- Set SaveDataInFormField
          oDocument.PrintDataInFormFields = true;
```

vb.net

```
'--- Return SaveDataInFormField
          Dim vSaveData As Boolean = oDocument.SaveDataInFormField

          '--- Set PrintDataInFormFields
          oDocument.SaveDataInFormField = True
```

Document.Sections

Description

Returns an array of Section objects that represents all the sections in this Document.

Signature: C#

```
public Section[] Sections{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Sections() As Section()
```

Document.SetVariable(String, String)

Description

Sets the value of the variable with the specified name. If the variable does not exist, `setVariable` creates a new variable. If the variable does exist, `setVariable` overwrites the existing variable.

Signature: C#

```
public void SetVariable(System.String name, System.String val)
```

Signature: VB.NET

```
Public Sub SetVariable(ByVal name As String, ByVal val As String)
```

Parameters

name

The name of the variable.

val

The value of the variable.

Examples

C#

```
doc.SetVariable("platform", "Windows");
```

vb.net

```
doc.SetVariable("platform", "Windows")
```

Document.Styles

Description

Returns a Styles object that represents the Styles embedded in this document.

Signature: C#

```
public Styles Styles{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Styles() As Styles
```

Remarks

MS Word equivalent: Format menu > Styles and Formatting...

Examples

C#

```
Styles style = oDocument.Styles;
```

vb.net

```
Dim styles As Styles = oDocument.Styles
```

Document.UseWidowControl

Description

Sets or returns a `boolean` representing if widow control is in effect for this Document.

Signature: C#

```
public boolean UseWidowControl{ get; set; }
```

Signature: VB.NET

```
Public Property UseWidowControl() As Boolean
```

Remarks

The default value for a new `Document` is `false`.

Examples

C#

```
//--- Return UseWidowControl
          bool useWidowControl = oDocument.UseWidowControl;

          //--- Set UseWidowControl
          oDocument.UseWidowControl = true;
```

vb.net

```
'--- Return UseWidowControl
          Dim useWidowControl As Boolean = oDocument.UseWidowControl

          '--- Set UseWidowControl
          oDocument.UseWidowControl = True
```

Document.ViewSettings

Description

Returns a WordViewSettings object that can be used to get and set various settings that pertain to how Word will display this document.

Signature: C#

```
public WordViewSettings ViewSettings{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ViewSettings() As WordViewSettings
```

Examples

C#

```
WordViewSettings oWordViewSettings = oDocument.ViewSettings;
```

vb.net

```
Dim oWordViewSettings As WordViewSettings = oDocument.ViewSettings
```

Document.Watermark

Description

Returns a Watermark object representing an image or text that appears in the background of a printed document.

Signature: C#

```
public Watermark Watermark{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Watermark() As Watermark
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
```

DocumentProperties

Description

This class contains all file properties that are used by Word and Windows Explorer to display and store metadata about documents. None of these properties will affect the content of a document.

Signature: C#

```
public sealed class DocumentProperties
```

Signature: VB.NET

```
Public NotInheritable Class DocumentProperties
```

Remarks

You cannot create a DocumentProperties object using the `new` keyword. You must get an existing `DocumentProperties` object from a Document.

MS Word equivalent: File menu > Properties

The following sample gets the DocumentProperties of a document so you can start working with these properties.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          DocumentProperties docProps = doc.DocumentProperties;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim docProps As DocumentProperties = doc.DocumentProperties
```

Properties

Name	Description
Application	Returns a `String` that is the name of the application that created this file.
Author	Sets or returns a `String` that is the name of the author that created this file.
Comments	Sets or returns a `String` that represents the comments attached to this file.
Company	Sets or returns a `String` that represents any company info added to this file.
CreationDate	Sets or returns an object that represents the creation date of this file.
Keywords	Sets or returns a `String` that represents any keywords that are added to this file.
LastSavedDate	Sets or returns a DateTime that represents the date this file was last saved.
Manager	Sets or returns a `String` that represents the name of the manager in charge of this file.
Subject	Sets or returns a `String` that represents the subject of this file.
Title	Sets or returns a `String` that is the title of this file.

Methods

Name	Description
Clear()	Clears all the Document properties for this document.
GetCustomProperty(String)	Returns an `Object` that represents the value of the custom property specified.
GetProperty(DocumentProperties.BuiltInProperty)	Returns an `Object` that represents the value of the property specified.
GetProperty(String)	Returns an `Object` that represents the value of the property specified.
RestoreDefaults()	Sets all the Document properties for this document to their default values.
SetCustomProperty(String, Object)	Sets the specified custom property with a certain value.
SetProperty(DocumentProperties.BuiltInProperty, Object)	Sets the value of the specified property.
SetProperty(String, Object)	Sets the value of the specified property.

Nested Classes

Name	Description
BuiltInProperty	`DocumentProperties.BuiltInProperty` contains a set of built-in document property fields.

DocumentProperties.Application

Description

Returns a `String` that is the name of the application that created this file.

Signature: C#

```
public System.String Application{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Application() As String
```

Examples

C#

```
string application = docProps.Application;
```

vb.net

```
Dim application As String = docProps.Application
```

DocumentProperties.Author

Description

Sets or returns a `String` that is the name of the author that created this file.

Signature: C#

```
public System.String Author{ get; set; }
```

Signature: VB.NET

```
Public Property Author() As String
```

Remarks

MS Word equivalent: File menu > Properties > Summary tab > Author:

Examples

C#

```
//--- Return Author
          string author = docProps.Author;

          //--- Set Author
          docProps.Author = "John Doe";
```

vb.net

```
'--- Return Author
          Dim author As String = docProps.Author

          '--- Set Author
          docProps.Author = "John Doe"
```

DocumentProperties.BuiltInProperty

Description

`DocumentProperties.BuiltInProperty` contains a set of built-in document property fields.

Signature: C#

```
public sealed class BuiltInProperty : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BuiltInProperty
  Inherits TypesafeEnum
```

Values

Name	Description
Application	The application used to create the document
Author	Author
Category	Category
Comments	Comments
Company	Company
CreationDate	Date the document was created
EditTime	Edit time
Keywords	Search keywords
LastPrintedDate	Last date the document was printed
LastSavedDate	Date the document was last saved
LastSavedBy	User that saved the document last
LinksUpToDate	Whether or not links in the document are up-to-date
Manager	Manager
Notes	Notes
RevisionNumber	Revision number
ScaleOrCrop	Scale or crop
Subject	Subject
Title	Title

DocumentProperties.Clear()

Description

Clears all the Document properties for this document.

Signature: C#

```
public void Clear()
```

Signature: VB.NET

```
Public Sub Clear()
```

Examples

C#

```
docProps.Clear();
```

vb.net

```
docProps.Clear()
```

DocumentProperties.Comments

Description

Sets or returns a `String` that represents the comments attached to this file.

Signature: C#

```
public System.String Comments{ get; set; }
```

Signature: VB.NET

```
Public Property Comments() As String
```

Remarks

MS Word equivalent: File menu > Properties > Summary tab > Comments:

Examples

C#

```
//--- Return Comments
          string comments = docProps.Comments;

          //--- Set Comments
          docProps.Comments = "This is for our next presentation.";
```

vb.net

```
'--- Return Comments
          Dim comments As String = docProps.Comments

          '--- Set Comments
          docProps.Comments = "This is for our next presentation."
```

DocumentProperties.Company

Description

Sets or returns a `String` that represents any company info added to this file.

Signature: C#

```
public System.String Company{ get; set; }
```

Signature: VB.NET

```
Public Property Company() As String
```

Remarks

MS Word equivalent: File menu > Properties > Summary tab > Company:

Examples

C#

```
//--- Return Company
          string company = docProps.Company;

          //--- Set Company
          docProps.Company = "SoftArtisans, Inc.";
```

vb.net

```
'--- Return Company
          Dim company As String = docProps.Company

          '--- Set Company
          docProps.Company = "SoftArtisans, Inc."
```

DocumentProperties.CreationDate

Description

Sets or returns an object that represents the creation date of this file.

Signature: C#

```
public System.DateTime CreationDate{ get; set; }
```

Signature: VB.NET

```
Public Property CreationDate() As Date
```

Remarks

MS Word equivalent: File menu > Properties > General tab > Created:

Examples

C#

```
//--- Return CreationDate
          DateTime oDateTime = docProps.CreationDate;

          //--- Set CreationDate
          docProps.CreationDate = DateTime.Now;
```

vb.net

```
'--- Return CreationDate
          Dim oDateTime As DateTime = docProps.CreationDate

          '--- Set CreationDate
          docProps.DateTime = DateTime.Now
```

DocumentProperties.GetCustomProperty(String)

Description

Returns an `Object` that represents the value of the custom property specified.

Signature: C#

```
public System.Object GetCustomProperty(System.String key)
```

Signature: VB.NET

```
Public Function GetCustomProperty(ByVal key As String) As Object
```

Parameters

key

The name of the property to retrieve.

Returns

An `Object` that represents the value of the custom property specified.

Remarks

MS Word equivalent: File menu > Properties > Custom tab > Properties:

Examples

C#

```
//--- Return "Priority" Custom Property
          string priority = docProps.GetCustomProperty("Priority");
```

vb.net

```
'--- Return "Priority" Custom Property
          Dim priority As String = docProps.GetCustomProperty("Priority")
```

DocumentProperties.GetProperty

Overload	Description

GetProperty(DocumentProperties.BuiltInProperty)	Returns an `Object` that represents the value of the property specified.
GetProperty(String)	Returns an `Object` that represents the value of the property specified.

DocumentProperties.GetProperty(DocumentProperties.BuiltInProperty)

Description

Returns an `Object` that represents the value of the property specified.

Signature: C#

```
public System.Object GetProperty(BuiltInProperty property)
```

Signature: VB.NET

```
Public Function GetProperty(ByVal [property] As BuiltInProperty) As Object
```

Parameters

property

A BuiltInProperty object that represents the built-in property to set.

Returns

An `Object` that represents the value of the property specified.

Remarks

MS Word equivalent: File menu > Properties > Custom tab > Properties:

Examples

C#

```
//--- Return "Author" Property
          string priority =
               (string)docProps.GetProperty(DocumentProperties.BuiltInProperty.Author);
```

vb.net

```
'--- Return "Author" Property
          Dim priority As String = _
               docProps.GetProperty(DocumentProperties.BuiltInProperty.Author)
```

DocumentProperties.GetProperty(String)

Description

Returns an `Object` that represents the value of the property specified.

Signature: C#

```
public System.Object GetProperty(System.String propertyName)
```

Signature: VB.NET

```
Public Function GetProperty(ByVal propertyName As String) As Object
```

Parameters

propertyName

A `String` representing the name of the property to retrieve.

Returns

An `Object` that represents the value of the property specified.

Remarks

MS Word equivalent: File menu > Properties > Custom tab > Properties:

Examples

C#

```
//--- Return "Priority" Property
          string priority = docProps.GetProperty("Priority");
```

vb.net

```
'--- Return "Priority" Property
          Dim priority As String = docProps.GetProperty("Priority")
```

DocumentProperties.Keywords

Description

Sets or returns a `String` that represents any keywords that are added to this file.

Signature: C#

```
public System.String Keywords{ get; set; }
```

Signature: VB.NET

```
Public Property Keywords() As String
```

Remarks

MS Word equivalent: File menu > Properties > Summary tab > Keywords:

Examples

C#

```
//--- Return Keywords
          string keywords = docProps.Keywords;

          //--- Set Keywords
          docProps.Keywords = "Confidential, Memo";
```

vb.net

```
'--- Return Keywords
          Dim keywords As String = docProps.Keywords

          '--- Set Keywords
          docProps.Keywords = "Confidential, Memo"
```

DocumentProperties.LastSavedDate

Description

Sets or returns a DateTime that represents the date this file was last saved.

Signature: C#

```
public System.DateTime LastSavedDate{ get; set; }
```

Signature: VB.NET

```
Public Property LastSavedDate() As Date
```

Examples

C#

```
//--- Return LastSavedDate
          DateTime oDateTime = docProps.LastSavedDate;

          //--- Set LastSavedDate
          docProps.LastSavedDate = DateTime.Now;
```

vb.net

```
'--- Return LastSavedDate
          Dim oDateTime As DateTime = docProps.LastSavedDate

          '--- Set LastSavedDate
          docProps.LastSavedDate = DateTime.Now
```

DocumentProperties.Manager

Description

Sets or returns a `String` that represents the name of the manager in charge of this file.

Signature: C#

```
public System.String Manager{ get; set; }
```

Signature: VB.NET

```
Public Property Manager() As String
```

Remarks

MS Word equivalent: File menu > Properties > Summary tab > Manager:

Examples

C#

```
//--- Return Manager
          string manager = docProps.Manager;

          //--- Set Manager
          docProps.Manager = "John Doe";
```

vb.net

```
'--- Return Manager
          Dim manager As String = docProps.Manager

          '--- Set Manager
          docProps.Manager = "John Doe"
```

DocumentProperties.RestoreDefaults()

Description

Sets all the Document properties for this document to their default values.

Signature: C#

```
public void RestoreDefaults()
```

Signature: VB.NET

```
Public Sub RestoreDefaults()
```

Examples

C#

```
docProps.RestoreDefaults();
```

vb.net

```
docProps.RestoreDefaults()
```

DocumentProperties.SetCustomProperty(String, Object)

Description

Sets the specified custom property with a certain value.

Signature: C#

```
public void SetCustomProperty(System.String key, System.Object value)
```

Signature: VB.NET

```
Public Sub SetCustomProperty(ByVal key As String, ByVal value As Object)
```

Parameters

key

A `String` that represents name of the property to set.

value

An `Object` that represents the value of the custom property specified.

Remarks

MS Word equivalent: File menu > Properties > Custom tab

Examples

C#

```
//--- Set "Priority" Custom Property
          docProps.SetCustomProperty("Priority", "High");
```

vb.net

```
'--- Set "Priority" Custom Property
          docProps.SetCustomProperty("Priority", "High")
```

DocumentProperties.SetProperty

Overload	Description
SetProperty(DocumentProperties.BuiltInProperty, Object)	Sets the value of the specified property.
SetProperty(String, Object)	Sets the value of the specified property.

DocumentProperties.SetProperty(DocumentProperties.BuiltInProperty, Object)

Description

Sets the value of the specified property.

Signature: C#

```
public void SetProperty(BuiltInProperty property, System.Object propertyValue)
```

Signature: VB.NET

```
Public Sub SetProperty(ByVal [property] As BuiltInProperty, ByVal propertyValue As Object)
```

Parameters

property

A DocumentProperties.BuiltInProperty object that represents the built-in property to set.

propertyValue

An `Object` that represents the value of the property specified.

Remarks

MS Word equivalent: File menu > Properties > Custom tab > Properties:

Examples

C#

```
//--- Set "Author" Property
          docProps.SetProperty(DocumentProperties.BuiltInProperty.Author, "Fred Jones");
```

vb.net

```
'--- Set "Priority" Property
          docProps.SetProperty(DocumentProperties.BuiltInProperty.Author, "Fred Jones")
```

DocumentProperties.SetProperty(String, Object)

Description

Sets the value of the specified property.

Signature: C#

```
public void SetProperty(System.String propertyName, System.Object propertyValue)
```

Signature: VB.NET

```
Public Sub SetProperty(ByVal propertyName As String, ByVal propertyValue As Object)
```

Parameters

propertyName

A `String` representing the name of the property to set.

propertyValue

An `Object` that represents the value of the property specified.

Remarks

MS Word equivalent: File menu > Properties > Custom tab > Properties:

Examples

C#

```
//--- Set "Priority" Property
          docProps.SetProperty("Priority", "high");
```

vb.net

```
'--- Set "Priority" Property
          docProps.SetProperty("Priority", "high")
```

DocumentProperties.Subject

Description

Sets or returns a `String` that represents the subject of this file.

Signature: C#

```
public System.String Subject{ get; set; }
```

Signature: VB.NET

```
Public Property Subject() As String
```

Remarks

MS Word equivalent: File menu > Properties > Summary tab > Subject:

Examples

C#

```
//--- Return Subject
          string subject = docProps.Subject;

          //--- Set Subject
          docProps.Subject = "Annual Meeting";
```

vb.net

```
'--- Return Subject
          Dim subject As String = docProps.Subject

          '--- Set Subject
          docProps.Subject = "Annual Meeting"
```

DocumentProperties.Title

Description

Sets or returns a `String` that is the title of this file.

Signature: C#

```
public System.String Title{ get; set; }
```

Signature: VB.NET

```
Public Property Title() As String
```

Remarks

MS Word equivalent: File menu > Properties > Summary tab > Title:

Examples

C#

```
//--- Return Title
          string title = docProps.Title;

          //--- Set Title
          docProps.Title = "Sample Word Document";
```

vb.net

```
'--- Return Author
          Dim title As String = docProps.Title

          '--- Set Author
          docProps.Title = "Sample Word Document"
```

DocumentProtection

Description

This class is used to get and set document protection settings for a document.

Signature: C#

```
public sealed class DocumentProtection
```

Signature: VB.NET

```
Public NotInheritable Class DocumentProtection
```

Remarks

You cannot create a DocumentProtection object using the `new` keyword. You must get an existing `DocumentProtection` object from a Document.

MS Word equivalent: Tools menu > Protect Document...

The following sample gets the `DocumentProtection` object of a document so you can start working with these properties.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          DocumentProtection docProtect = doc.DocumentProtection;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim docProtect As DocumentProtection = doc.DocumentProtection
```

Properties

Name	Description
AnnotationsLocked	Sets or returns a `boolean` representing whether or not annotations are locked in this document.
AutoFormatOverride	Sets or returns a `boolean` representing whether or not Word will autoformat to override a style lock down in this document.
DocumentProtected	Sets or returns a protection level for the document. Possible values are: None, TrackChanges, Comments, Forms, and ReadOnly.
EnforceDocumentProtection	Sets or returns a `boolean` representing whether or not Word will enforce document protection in this document.
SelectOnlyFormFields	Sets or returns a `boolean` representing whether or not Word will only allow selections in form fields for this document.
StylesLocked	Sets or returns a `boolean` representing whether or not Word will lock the Styles in this document.

Nested Classes

Name	Description
ProtectionLevel	A DocumentProtection.ProtectionLevel value specifies what type of editing is allowed in the document.

DocumentProtection.AnnotationsLocked

Description

Sets or returns a `boolean` representing whether or not annotations are locked in this document.

Signature: C#

```
public boolean AnnotationsLocked{ get; set; }
```

Signature: VB.NET

```
Public Property AnnotationsLocked() As Boolean
```

Examples

C#

```
//--- Get AnnotationsLocked
          bool annotationsLocked = docProtect.AnnotationsLocked;

          //--- Set AnnotationsLocked
          docProtect.AnnotationsLocked = true;
```

vb.net

```
'--- Get AnnotationsLocked
          Dim annotationsLocked As Boolean = docProtect.AnnotationsLocked

          '--- Set AnnotationsLocked
          docProtect.AnnotationsLocked = True
```

DocumentProtection.AutoFormatOverride

Description

Sets or returns a `boolean` representing whether or not Word will autoformat to override a style lock down in this document.

Signature: C#

```
public boolean AutoFormatOverride{ get; set; }
```

Signature: VB.NET

```
Public Property AutoFormatOverride() As Boolean
```

Remarks

MS Word equivalent: Tools menu > Protect Document... > 1. Formatting restrictions section > Settings... link > Formatting section > Allow AutoFormat to override formatting restrictions

Examples

C#

```
//---- Get AutoFormatOverride
          bool autoFormatOverride = docProtect.AutoFormatOverride;

          //--- Set AutoFormatOverride
          docProtect.AutoFormatOverride = true;
```

vb.net

```
'--- Get AutoFormatOverride
          Dim autoFormatOverride As Boolean = docProtect.AutoFormatOverride

          '--- Set AutoFormatOverride
          docProtect.AutoFormatOverride = True
```

DocumentProtection.DocumentProtected

Description

Sets or returns a protection level for the document. Possible values are: None, TrackChanges, Comments, Forms, and ReadOnly.

Signature: C#

```
public ProtectionLevel DocumentProtected{ get; set; }
```

Signature: VB.NET

```
Public Property DocumentProtected() As ProtectionLevel
```

Examples

C#

```
//--- Get ProtectionLevel
          DocumentProtection.ProtectionLevel level = docProtect.DocumentProtected;

          //--- Set ProtectionLevel to allow editing of Forms only
          docProtect.DocumentProtected = DocumentProtection.ProtectionLevel.Comments;
```

vb.net

```
'--- Get ProtectionLevel
          Dim level As DocumentProtection.ProtectionLevel = docProtect.DocumentProtected

          '--- Set ProtectionLevel to allow editing of Forms only
          docProtect.DocumentProtected = DocumentProtection.ProtectionLevel.Comments
```

DocumentProtection.EnforceDocumentProtection

Description

Sets or returns a `boolean` representing whether or not Word will enforce document protection in this document.

Signature: C#

```
public boolean EnforceDocumentProtection{ get; set; }
```

Signature: VB.NET

```
Public Property EnforceDocumentProtection() As Boolean
```

Remarks

MS Word equivalent: Protect = Tools menu > Protect Document... > 3. Start enforcement section > Yes, Start Enforcing Protection button & Unprotect = Tools menu > Unprotect Document

Examples

C#

```
//--- Get EnforceDocumentProtection
          bool enforceDocumentProtection = docProtect.EnforceDocumentProtection;

          //--- Set EnforceDocumentProtection
          docProtect.EnforceDocumentProtection = true;
```

vb.net

```
'--- Get EnforceDocumentProtection
          Dim enforceDocumentProtection As Boolean = docProtect.EnforceDocumentProtection

          '--- Set EnforceDocumentProtection
          docProtect.EnforceDocumentProtection = True
```

DocumentProtection.ProtectionLevel

Description

A DocumentProtection.ProtectionLevel value specifies what type of editing is allowed in the document.

Signature: C#

```
public sealed class ProtectionLevel : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ProtectionLevel
  Inherits TypesafeEnum
```

Values

Name	Description
Comments	Only comments may be edited.
Forms	Only form fields may be modified.
None	All parts of the document may be modified.
ReadOnly	The document cannot be modified.
TrackChanges	Word's change tracking feature will be enabled.

DocumentProtection.SelectOnlyFormFields

Description

Sets or returns a `boolean` representing whether or not Word will only allow selections in form fields for this document.

Signature: C#

```
public boolean SelectOnlyFormFields{ get; set; }
```

Signature: VB.NET

```
Public Property SelectOnlyFormFields() As Boolean
```

Examples

C#

```
//--- Get SelectOnlyFormFields
          bool selectOnlyFormFields = docProtect.SelectOnlyFormFields;

          //--- Set SelectOnlyFormFields
          docProtect.SelectOnlyFormFields = true;
```

vb.net

```
'--- Get SelectOnlyFormFields
          Dim selectOnlyFormFields As Boolean = docProtect.SelectOnlyFormFields

          '--- Set SelectOnlyFormFields
          docProtect.SelectOnlyFormFields = True
```

DocumentProtection.StylesLocked

Description

Sets or returns a `boolean` representing whether or not Word will lock the Styles in this document.

Signature: C#

```
public boolean StylesLocked{ get; set; }
```

Signature: VB.NET

```
Public Property StylesLocked() As Boolean
```

Examples

C#

```
//--- Get StylesLocked
          bool stylesLocked = docProtect.StylesLocked;

          //--- Set StylesLocked
          oDocumentProperties.StylesLocked = true;
```

vb.net

```
'--- Get StylesLocked
          Dim stylesLocked As Boolean = docProtect.StylesLocked

          '--- Set StylesLocked
          docProtect.StylesLocked = True
```

Element

Description

This class is used to represent a region of the document that can be edited. This is the main class for manipulating the Word object model. Most elements of a Document inherit from the Element class.

Signature: C#

```
public class Element
```

Signature: VB.NET

```
Public Class Element
```

Remarks

You cannot create a new `Element` object by using the `new` keyword. There are various properties

The following sample demonstrates getting an `Element` object by retrieving a section's header:

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Section sect = doc.Sections[0];
          Element e = sect.GetHeader(Section.HeaderFooterType.All);
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim sect As Section = doc.Sections(0)
          Dim e As Element = sect.GetHeader(Section.HeaderFooterType.All)
```

Properties

Name	Description
Children	Returns a collection of Element objects that represents all the direct children of the current element.
ElementType	Returns an Element.Type object that represents the type of this element.
Length	Returns an `int` that represents the number of characters in this Element.
Start	Returns an `int` that represents the absolute character index of the start of this Element.
Text	Returns a `String` that represents the text for this Element.

Methods

Name	Description
CreateBookmark(String)	Creates a bookmark on this Element with the specified name.
CreateSectionAfter()	Creates and returns a Section object that represents a new section at the end of this Element.
CreateSectionBefore()	Creates and returns a Section object that represents a new section at the beginning of this Element.
DeleteElement()	Deletes all the text and properties of this Element. The element then is no longer a valid element of the type. It is now just a position of where that element used to be.
GetElements(Element.Type)	Returns a collection of Element objects that represent all the elements in the document of the specified type.
GetPosition(Int32)	Returns a Position object that represents the location specified by the offset. The Position object can be used similar to a cursor.
ImportData(Object()())	Imports the data from a 2-dimensional array into a Table at the point of the current element in the document.

ImportData(Object()(), String(), DataImportProperties)	Imports a set of data from an array of column names and a 2-dimensional array of data to a Table created at the point of the current element in the document. Also accepts a DataImportProperties object defining any options for this data import.
ImportData(System.Data.DataTable)	Imports data from a `DataTable` into a Table at the point of this Element in the document.
ImportData(System.Data.DataTable, DataImportProperties)	Imports data from a `DataTable` into a Table at the point of this Element in the document. Also accepts a DataImportProperties object that contains the options for importing.
ImportData(System.Data.DataView)	Imports data from a `DataView` into a Table at the point of this Element in the document.
ImportData(System.Data.DataView, DataImportProperties)	Imports data from a `DataView` into a Table at the point of this Element in the document. Also accepts a DataImportProperties object that contains the options for importing.
ImportData(Object(,))	Imports the data from a rectangular array into a Table at the point of the current element in the document.
ImportData(Object(,), String(), DataImportProperties)	Imports the data from a rectangular array into a Table at the point of the current element in the document. Also accepts an array of column names and a DataImportProperties object with the import options.
ImportData(System.Data.IDataReader)	Imports the data from a DataReader into a Table at the point of the current element in the document.
ImportData(System.Data.IDataReader, DataImportProperties)	Imports the data from a DataReader into a Table at the point of the current element in the document. Also accepts a DataImportProperties object that contains the import options.
InsertAfter(Element)	Inserts an existing element after this element. A copy of the Element passed in is placed after this element
InsertBefore(Element)	Inserts an existing element before this element. A copy of the Element passed in is placed before this element
InsertBreakAfter(Element.BreakType)	Inserts a break of the specified type after the element.
InsertBreakBefore(Element.BreakType)	Inserts a break of the specified type before the element.
InsertHyperlinkAfter(String, String)	Creates and returns a Hyperlink at the beginning of this Element. This hyperlink will have the specified url and displayed text.
InsertHyperlinkBefore(String, String)	Creates and returns a Hyperlink at the beginning of this Element. This hyperlink will have the specified url and displayed text.
InsertImageAfter(System.IO.Stream)	Inserts and returns an InlineImage at the end of this Element. The image that will be inserted is specified by the stream parameter.
InsertImageAfter(String)	Inserts and returns an InlineImage at the end of this Element. The image that will be inserted is specified by the fileName parameter.
InsertImageBefore(String)	Inserts and returns an InlineImage at the beginning of this Element. The image that will be inserted is specified by the fileName parameter.
InsertImageBefore(System.IO.Stream)	Inserts and returns an InlineImage at the beginning of this Element. The image that will be inserted is specified by the stream parameter.
InsertListAfter(Boolean)	Creates and returns an empty List at the end of this Element. By passing a boolean, it can be a numbered list (true) or a bulleted list (false).
InsertListBefore(Boolean)	Creates and returns an empty List at the beginning of this Element. By passing a boolean, it can be a numbered list (true) or a bulleted list (false).
InsertMergeFieldAfter(String, String)	Creates and returns a MergeField at the end of this Element. This merge field will have the specified name and contents.
InsertMergeFieldBefore(String, String)	Creates and returns a MergeField at the beginning of this Element. This merge field will have the specified name and contents.
InsertParagraphAfter(NamedStyle)	Creates and returns a Paragraph object that represents a new empty paragraph at the end of this Element. The paragraph that is inserted will have the style specified. If this Element is in the middle of a paragraph, the containing paragraph will be split.
InsertParagraphAfter(NamedStyle, ParagraphFormatting)	Creates and returns a Paragraph object that represents a new empty paragraph at the end of this Element. The paragraph that is inserted will have the style specified, to which any additional specified paragraph formatting is applied as well. If this Element is in the middle of a paragraph, the containing paragraph will be split.

InsertParagraphBefore(NamedStyle)	Creates and returns a Paragraph object that represents a new empty paragraph at the beginning of this Element. The paragraph that is inserted will have the style specified. If this Element is in the middle of a paragraph, the containing paragraph will be split.
InsertParagraphBefore(NamedStyle, ParagraphFormatting)	Creates and returns a Paragraph object that represents a new empty paragraph at the beginning of this Element. The paragraph that is inserted will have the style specified, to which any additional specified paragraph formatting is applied as well. If this Element is in the middle of a paragraph, the containing paragraph will be split.
InsertTableAfter(Int32, Int32)	Creates and returns a Table at the end of this Element. The table will contain the specified number of rows and columns.
InsertTableAfter(Int32, Int32, TableFormatting)	Creates and returns a Table at the end of this Element. The table will contain the specified number of rows and columns. It will also be formatted with the properties specified.
InsertTableBefore(Int32, Int32)	Creates and returns a Table at the beginning of this Element. The table will contain the specified number of rows and columns.
InsertTableBefore(Int32, Int32, TableFormatting)	Creates and returns a Table at the beginning of this Element. The table will contain the specified number of rows and columns. It will also be formatted with the properties specified.
InsertTextAfter(String, Boolean)	Creates and returns a CharacterRun at the end of this Element. The text to be inserted is specified as well as whether this should be a new character run or part of the previous one.
InsertTextAfter(String, Font)	Creates and returns a CharacterRun at the end of this Element. The text to be inserted is specified as well as the font that should override the font found in the style of the character run's enclosing paragraph.
InsertTextBefore(String, Boolean)	Creates and returns a CharacterRun at the beginning of this Element. The text to be inserted is specified as well as whether this should be a new character run or part of the existing one.
InsertTextBefore(String, Font)	Creates and returns a CharacterRun at the beginning of this Element. The text to be inserted is specified as well as the font that should override the font found in the style of the character run's enclosing paragraph.
Search(String)	Searches for a specified string and returns a collection of type SearchMatch[].
SearchAndReplace(String, String)	Searches for a string specified by the `search` parameter, and replaces it with the string specified by the `replaceWith` parameter. The search string can be a literal, a regular expression, or a combination of both. Since the string can contain regular expressions, use a backslash to escape special characters. Special characters are `[^$.|?*+()`.
SubElement(Int32, Int32)	Creates and returns a Element object that represents a sub Element within this Element based on the specified start and end offsets.

Extension Methods

Introduced in build 4.5

Overload	Description
ImportData(Microsoft.SharePoint.SPList)	Imports the data from a SharePoint List into a Table at the point of the current element in the document.
ImportData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)	Imports the data from a SharePoint View into a Table at the point of the current element in the document. Also accepts a SharePoint List to which the View belongs and a DataImportProperties object with the import options.

Nested Classes

Name	Description
BreakType	An Element.BreakType value specifies the type of break to insert Element.InsertBreakBefore or Element.InsertBreakAfter a region in the document.
Type	An Element represents a region of the document that can be edited. An Element.Type value specifies the type of region an `Element` object represents.

Element.BreakType

Description

An Element.BreakType value specifies the type of break to insert Element.InsertBreakBefore or Element.InsertBreakAfter a region in the document.

Signature: C#

```
public sealed class BreakType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BreakType
  Inherits TypesafeEnum
```

Values

Name	Description
Column	Column break
Page	Page
TextWrap	Text wrapping break

Element.Children

Description

Returns a collection of Element objects that represents all the direct children of the current element.

Signature: C#

```
public Element[] Children{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Children() As Element()
```

Examples

C#

```
Element[] eArray = e.Children;
```

vb.net

```
Dim eArray As Element() = e.Children
```

Element.CreateBookmark(String)

Description

Creates a bookmark on this Element with the specified name.

Signature: C#

```
public virtual void CreateBookmark(System.String name)
```

Signature: VB.NET

```
Public Overridable Sub CreateBookmark(ByVal name As String)
```

Parameters

name

A `String` representing the name to give this bookmark.

Examples

C#

```
e.CreateBookmark("MyBookmark");
```

vb.net

```
e.CreateBookmark("MyBookmark")
```

Element.CreateSectionAfter()

Description

Creates and returns a Section object that represents a new section at the end of this Element.

Signature: C#

```
public virtual Section CreateSectionAfter()
```

Signature: VB.NET

```
Public Overridable Function CreateSectionAfter() As Section
```

Returns

A `Section` object that represents a new section at the end of this Element.

Examples

C#

```
Section sect = e.CreateSectionAfter();
```

vb.net

```
Dim sect As Section = e.CreateSectionAfter()
```

Element.CreateSectionBefore()

Description

Creates and returns a Section object that represents a new section at the beginning of this Element.

Signature: C#

```
public virtual Section CreateSectionBefore()
```

Signature: VB.NET

```
Public Overridable Function CreateSectionBefore() As Section
```

Returns

A `Section` object that represents a new section at the beginning of this Element.

Examples

C#

```
Section sect = e.CreateSectionBefore();
```

vb.net

```
Dim sect As Section = e.CreateSectionBefore()
```

Element.DeleteElement()

Description

Deletes all the text and properties of this Element. The element then is no longer a valid element of the type. It is now just a position of where that element used to be.

Signature: C#

```
public virtual void DeleteElement()
```

Signature: VB.NET

```
Public Overridable Sub DeleteElement()
```

Examples

C#

```
e.DeleteElement();
```

vb.net

```
e.DeleteElement()
```

Element.ElementType

Description

Returns an Element.Type object that represents the type of this element.

Signature: C#

```
public Type ElementType{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ElementType() As Type
```

Element.GetElements(Element.Type)

Description

Returns a collection of Element objects that represent all the elements in the document of the specified type.

Signature: C#

```
public virtual Element[] GetElements(Type type)
```

Signature: VB.NET

```
Public Overridable Function GetElements(ByVal type As Type) As Element()
```

Parameters

type

An Element.Type representing the type of element to be returned.

Returns

A a collection of `Element` objects that represent all the elements in the document of the specified type.

Examples

C#

```
Element[] eArray = e.GetElements(Element.Type.Paragraph);
```

vb.net

```
Dim eArray As Element() = e.GetElements(Element.Type.Paragraph)
```

Element.GetPosition(Int32)

Description

Returns a Position object that represents the location specified by the offset. The Position object can be used similar to a cursor.

Signature: C#

```
public virtual Position GetPosition(int offset)
```

Signature: VB.NET

```
Public Overridable Function GetPosition(ByVal offset As Integer) As Position
```

Parameters

offset

An `int` representing the relative offset from the start of this Element to get the Position for.

Returns

A `Position` object that represents the location specified by the offset.

Examples

C#

```
Position oPosition = e.GetPosition();
```

vb.net

```
Dim oPosition As Position = e.GetPosition()
```

Element.ImportData

Overload	Description
ImportData(Object()())	Imports the data from a 2-dimensional array into a Table at the point of the current element in the document.
ImportData(Object()(), String(), DataImportProperties)	Imports a set of data from an array of column names and a 2-dimensional array of data to a Table created at the point of the current element in the document. Also accepts a DataImportProperties object defining any options for this data import.
ImportData(System.Data.DataTable)	Imports data from a `DataTable` into a Table at the point of this Element in the document.
ImportData(System.Data.DataTable, DataImportProperties)	Imports data from a `DataTable` into a Table at the point of this Element in the document. Also accepts a DataImportProperties object that contains the options for importing.
ImportData(System.Data.DataView)	Imports data from a `DataView` into a Table at the point of this Element in the document.
ImportData(System.Data.DataView, DataImportProperties)	Imports data from a `DataView` into a Table at the point of this Element in the document. Also accepts a DataImportProperties object that contains the options for importing.
ImportData(Object(,))	Imports the data from a rectangular array into a Table at the point of the current element in the document.
ImportData(Object(,), String(), DataImportProperties)	Imports the data from a rectangular array into a Table at the point of the current element in the document. Also accepts an array of column names and a DataImportProperties object with the import options.
ImportData(System.Data.IDataReader)	Imports the data from a DataReader into a Table at the point of the current element in the document.
ImportData(System.Data.IDataReader, DataImportProperties)	Imports the data from a DataReader into a Table at the point of the current element in the document. Also accepts a DataImportProperties object that contains the import options.

SharePoint Integration Extension Methods

Overload	Description
ImportData(Microsoft.SharePoint.SPList)	Imports the data from a SharePoint List into a Table at the point of the current element in the document.
ImportData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)	Imports the data from a SharePoint View into a Table at the point of the current element in the document. Also accepts a SharePoint List to which the View belongs and a DataImportProperties object with the import options.

Element.ImportData(Object(,), String(), DataImportProperties)

Description

Imports the data from a rectangular array into a Table at the point of the current element in the document. Also accepts an array of column names and a DataImportProperties object with the import options.

Signature: C#

```
public Table ImportData(System.Object[,] data, System.String[] columnNames, DataImportProperties
props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object(,), ByVal columnNames As String(), ByVal props As
DataImportProperties) As Table
```

Parameters

data

The rectangular array to import the data from.

columnNames

Array of column names, corresponding to columns of data in the rectangular array.

props

Import properties for this data import, including max rows, max columns, column filter, etc.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(rectArray, colArray, importProps);
```

vb.net

```
Dim tbl As Table = e.ImportData(rectArray, colArray, importProps)
```

Element.ImportData(Object(,))

Description

Imports the data from a rectangular array into a Table at the point of the current element in the document.

Signature: C#

```
public Table ImportData(System.Object[,] data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object(,)) As Table
```

Parameters

data

The rectangular array to import the data from.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(rectArray);
```

vb.net

```
Dim tbl As Table = e.ImportData(rectArray)
```

Element.ImportData(Object()(), String(), DataImportProperties)

Description

Imports a set of data from an array of column names and a 2-dimensional array of data to a Table created at the point of the current element in the document. Also accepts a DataImportProperties object defining any options for this data import.

Signature: C#

```
public virtual Table ImportData(System.Object[][] data, System.String[] columnNames,
DataImportProperties properties)
```

Signature: VB.NET

```
Public Overridable Function ImportData(ByVal data As Object()(), ByVal columnNames As String(), ByVal
properties As DataImportProperties) As Table
```

Parameters

data

A 2-dimensional array from which to import the data from.

columnNames

An array of column names to import.

properties

Import properties for this data import, including max rows, max columns, column filter, etc.

Returns

The `Table` that the data was imported into.

Examples

C#
`Table tbl = e.ImportData(dataArray, colArray, oDataImportProperties);`

vb.net
`Dim tbl As Table = e.ImportData(dataArray, colArray, oDataImportProperties)`

Element.ImportData(Object()())

Description

Imports the data from a 2-dimensional array into a Table at the point of the current element in the document.

Signature: C#

```
public virtual Table ImportData(System.Object[][] data)
```

Signature: VB.NET

```
Public Overridable Function ImportData(ByVal data As Object()()) As Table
```

Parameters

data

A 2-dimensional array from which to import the data from.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(dataArray);
```

vb.net

```
Dim tbl As Table = e.ImportData(dataArray)
```

Element.ImportData(System.Data.DataTable, DataImportProperties)

Description

Imports data from a `DataTable` into a Table at the point of this Element in the document. Also accepts a DataImportProperties object that contains the options for importing.

Signature: C#

```
public Table ImportData(System.Data.DataTable data, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataTable, ByVal props As DataImportProperties)
As Table
```

Parameters

data

Contains the data to be imported.

props

Import properties for this data import, including max rows, max columns, column filter, etc.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(dt, importProps);
```

vb.net

```
Dim tbl As Table = e.ImportData(dt, importProps)
```

Element.ImportData(System.Data.DataTable)

Description

Imports data from a `DataTable` into a Table at the point of this Element in the document.

Signature: C#

```
public Table ImportData(System.Data.DataTable data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataTable) As Table
```

Parameters

data

Contains the data to be imported.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(dt);
```

vb.net

```
Dim tbl As Table = e.ImportData(dt)
```

Element.ImportData(System.Data.DataView, DataImportProperties)

Description

Imports data from a `DataView` into a Table at the point of this Element in the document. Also accepts a DataImportProperties object that contains the options for importing.

Signature: C#

```
public Table ImportData(System.Data.DataView data, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataView, ByVal props As DataImportProperties) As
Table
```

Parameters

data

Contains the data to be imported.

props

Import properties for this data import, including max rows, max columns, column filter, etc.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(dv, importProps);
```

vb.net

```
Dim tbl As Table = e.ImportData(dv, importProps)
```

Element.ImportData(System.Data.DataView)

Description

Imports data from a `DataView` into a Table at the point of this Element in the document.

Signature: C#

```
public Table ImportData(System.Data.DataView data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataView) As Table
```

Parameters

data

Contains the data to be imported.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(dv);
```

vb.net

```
Dim tbl As Table = e.ImportData(dv)
```

Element.ImportData(System.Data.IDataReader, DataImportProperties)

Description

Imports the data from a DataReader into a Table at the point of the current element in the document. Also accepts a DataImportProperties object that contains the import options.

Signature: C#

```
public Table ImportData(System.Data.IDataReader data, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.IDataReader, ByVal props As DataImportProperties)
As Table
```

Parameters

data

DataReader to import the data from.

props

Import properties for this data import, including max rows, max columns, column filter, etc.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(dr, importProps);
```

vb.net

```
Dim tbl As table = e.ImportData(dr, importProps)
```

Element.ImportData(System.Data.IDataReader)

Description

Imports the data from a DataReader into a Table at the point of the current element in the document.

Signature: C#

```
public Table ImportData(System.Data.IDataReader data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.IDataReader) As Table
```

Parameters

data

DataReader to import the data from.

Returns

The `Table` that the data was imported into.

Examples

C#

```
Table tbl = e.ImportData(dr);
```

vb.net

```
Dim tbl As Table = e.ImportData(dr)
```

Element.InsertAfter(Element)

Description

Inserts an existing element after this element. A copy of the Element passed in is placed after this element

Signature: C#

```
public virtual void InsertAfter(Element element)
```

Signature: VB.NET

```
Public Overridable Sub InsertAfter(ByVal element As Element)
```

Parameters

element

An Element object to insert.

Element.InsertBefore(Element)

Description

Inserts an existing element before this element. A copy of the Element passed in is placed before this element

Signature: C#

```
public virtual void InsertBefore(Element element)
```

Signature: VB.NET

```
Public Overridable Sub InsertBefore(ByVal element As Element)
```

Parameters

element

An Element object to insert.

Element.InsertBreakAfter(Element.BreakType)

Description

Inserts a break of the specified type after the element.

Signature: C#

```
public virtual void InsertBreakAfter(BreakType type)
```

Signature: VB.NET

```
Public Overridable Sub InsertBreakAfter(ByVal type As BreakType)
```

Parameters

type

A Element.BreakType that specifies the type of break to insert

Element.InsertBreakBefore(Element.BreakType)

Description

Inserts a break of the specified type before the element.

Signature: C#

```
public virtual void InsertBreakBefore(BreakType type)
```

Signature: VB.NET

```
Public Overridable Sub InsertBreakBefore(ByVal type As BreakType)
```

Parameters

type

A Element.BreakType that specifies the type of break to insert

Element.InsertHyperlinkAfter(String, String)

Description

Creates and returns a Hyperlink at the beginning of this Element. This hyperlink will have the specified url and displayed text.

Signature: C#

```
public virtual Hyperlink InsertHyperlinkAfter(System.String url, System.String display)
```

Signature: VB.NET

```
Public Overridable Function InsertHyperlinkAfter(ByVal url As String, ByVal display As String) As
Hyperlink
```

Parameters

url

A `String` representing the url to link to.

display

A `String` representing the text that will displayed in the document for this hyperlink.

Returns

A `Hyperlink` object representing the newly created hyperlink.

Exceptions

System.Exception

Examples

C#

```
Hyperlink link = e.InsertHyperlinkAfter("http://www.softartisans.com", "SoftArtisans");
```

vb.net

```
Dim link As Hyperlink = e.InsertHyperlinkAfter("http://www.softartisans.com", "SoftArtisans")
```

Element.InsertHyperlinkBefore(String, String)

Description

Creates and returns a Hyperlink at the beginning of this Element. This hyperlink will have the specified url and displayed text.

Signature: C#

```
public virtual Hyperlink InsertHyperlinkBefore(System.String url, System.String display)
```

Signature: VB.NET

```
Public Overridable Function InsertHyperlinkBefore(ByVal url As String, ByVal display As String) As
Hyperlink
```

Parameters

url

A `String` representing the url to link to.

display

A `String` representing the text that will be displayed in the document for this hyperlink.

Returns

A `Hyperlink` object representing the newly created hyperlink.

Exceptions

System.Exception

Examples

C#

```
Hyperlink link = e.InsertHyperlinkBefore("http://www.softartisans.com", "SoftArtisans");
```

vb.net

```
Dim link As Hyperlink = e.InsertHyperlinkBefore("http://www.softartisans.com", "SoftArtisans")
```

Element.InsertImageAfter

Overload	Description
InsertImageAfter(String)	Inserts and returns an InlineImage at the end of this Element. The image that will be inserted is specified by the fileName parameter.
InsertImageAfter(System.IO.Stream)	Inserts and returns an InlineImage at the end of this Element. The image that will be inserted is specified by the stream parameter.

Element.InsertImageAfter(String)

Description

Inserts and returns an InlineImage at the end of this Element. The image that will be inserted is specified by the fileName parameter.

Signature: C#

```
public virtual InlineImage InsertImageAfter(System.String fileName)
```

Signature: VB.NET

```
Public Overridable Function InsertImageAfter(ByVal fileName As String) As InlineImage
```

Parameters

fileName

A `String` containing the filename of the image to insert.

Returns

An `InlineImage` object that represents the image that was inserted.

Exceptions

System.Exception

If the image file is not a valid format. See InlineImage.Format for a list of supported formats.

Examples

C#

```
InlineImage image = e.InsertImageAfter("C:\\sample.jpg");
```

vb.net

```
Dim image As InlineImage = e.InsertImageAfter("C:\sample.jpg")
```

Element.InsertImageAfter(System.IO.Stream)

Description

Inserts and returns an InlineImage at the end of this Element. The image that will be inserted is specified by the stream parameter.

Signature: C#

```
public virtual InlineImage InsertImageAfter(System.IO.Stream stream)
```

Signature: VB.NET

```
Public Overridable Function InsertImageAfter(ByVal stream As System.IO.Stream) As InlineImage
```

Parameters

stream

A `Stream` containing the image to insert.

Returns

An `InlineImage` object that represents the image that was inserted.

Exceptions

System.Exception

If the image file is not a valid format. See InlineImage.Format for a list of supported formats.

Examples

C#

```
InlineImage image = e.InsertImageAfter(strm);
```

vb.net

```
Dim image As InlineImage = e.InsertImageAfter(strm)
```

Element.InsertImageBefore

Overload	Description
InsertImageBefore(String)	Inserts and returns an InlineImage at the beginning of this Element. The image that will be inserted is specified by the fileName parameter.
InsertImageBefore(System.IO.Stream)	Inserts and returns an InlineImage at the beginning of this Element. The image that will be inserted is specified by the stream parameter.

Element.InsertImageBefore(String)

Description

Inserts and returns an InlineImage at the beginning of this Element. The image that will be inserted is specified by the fileName parameter.

Signature: C#

```
public virtual InlineImage InsertImageBefore(System.String fileName)
```

Signature: VB.NET

```
Public Overridable Function InsertImageBefore(ByVal fileName As String) As InlineImage
```

Parameters

fileName

A `String` containing the filename of the image to insert.

Returns

An `InlineImage` object that represents the image that was inserted.

Exceptions

System.Exception

If the image file is not a valid format. See InlineImage.Format for a list of supported formats.

Examples

C#

```
InlineImage image = e.InsertImageBefore("C:\sample.jpg");
```

vb.net

```
Dim image As InlineImage = e.InsertImageBefore("C:\sample.jpg")
```

Element.InsertImageBefore(System.IO.Stream)

Description

Inserts and returns an InlineImage at the beginning of this Element. The image that will be inserted is specified by the stream parameter.

Signature: C#

```
public virtual InlineImage InsertImageBefore(System.IO.Stream stream)
```

Signature: VB.NET

```
Public Overridable Function InsertImageBefore(ByVal stream As System.IO.Stream) As InlineImage
```

Parameters

stream

A `Stream` containing the image to insert. to insert.

Returns

An `InlineImage` object that represents the image that was inserted.

Exceptions

System.Exception

If the image file is not a valid format. See InlineImage.Format for a list of supported formats.

Examples

C#

```
InlineImage image = e.InsertImageBefore(strm);
```

vb.net

```
Dim image As InlineImage = e.InsertImageBefore(strm)
```

Element.InsertListAfter(Boolean)

Description

Creates and returns an empty List at the end of this Element. By passing a boolean, it can be a numbered list (true) or a bulleted list (false).

Signature: C#

```
public virtual List InsertListAfter(boolean numbered)
```

Signature: VB.NET

```
Public Overridable Function InsertListAfter(ByVal numbered As Boolean) As List
```

Parameters

numbered

Returns

A `List` object representing the newly created empty list.

Examples

C#

```
List lst = e.InsertListAfter(true);
```

vb.net

```
Dim lst As List = e.InsertListAfter(True)
```

Element.InsertListBefore(Boolean)

Description

Creates and returns an empty List at the beginning of this Element. By passing a boolean, it can be a numbered list (true) or a bulleted list (false).

Signature: C#

```
public virtual List InsertListBefore(boolean numbered)
```

Signature: VB.NET

```
Public Overridable Function InsertListBefore(ByVal numbered As Boolean) As List
```

Parameters

numbered

Returns

A `List` object representing the newly created empty list.

Examples

C#

```
List lst = e.InsertListBefore(true);
```

vb.net

```
Dim lst As List = e.InsertListBefore(True)
```

Element.InsertMergeFieldAfter(String, String)

Description

Creates and returns a MergeField at the end of this Element. This merge field will have the specified name and contents.

Signature: C#

```
public virtual MergeField InsertMergeFieldAfter(System.String fieldName, System.String display)
```

Signature: VB.NET

```
Public Overridable Function InsertMergeFieldAfter(ByVal fieldName As String, ByVal display As String)
As MergeField
```

Parameters

fieldName

A `String` representing the name of this merge field.

display

A `String` representing the text that will be displayed for this merge field in the document.

Returns

A `MergeField` object representing the newly created merge field.

Exceptions

System.Exception

Examples

C#

```
MergeField mfield = e.InsertMergeFieldAfter("MyField", "ThisNameIsDisplayed");
```

vb.net

```
Dim mfield As MergeField = e.InsertMergeFieldAfter("MyField", "ThisNameIsDisplayed")
```

Element.InsertMergeFieldBefore(String, String)

Description

Creates and returns a MergeField at the beginning of this Element. This merge field will have the specified name and contents.

Signature: C#

```
public virtual MergeField InsertMergeFieldBefore(System.String fieldName, System.String display)
```

Signature: VB.NET

```
Public Overridable Function InsertMergeFieldBefore(ByVal fieldName As String, ByVal display As String)
As MergeField
```

Parameters

fieldName

A `String` representing the name of this merge field.

display

A `String` representing the text that will displayed for this merge field in the document.

Returns

A `MergeField` object representing the newly created merge field.

Exceptions

System.Exception

Examples

C#

```
MergeField mfield = e.InsertMergeFieldBefore("MyField", "ThisNameIsDisplayed");
```

vb.net

```
Dim mfield As MergeField = e.InsertMergeFieldBefore("MyField", "ThisNameIsDisplayed")
```

Element.InsertParagraphAfter

Overload	Description
InsertParagraphAfter(NamedStyle)	Creates and returns a Paragraph object that represents a new empty paragraph at the end of this Element. The paragraph that is inserted will have the style specified. If this Element is in the middle of a paragraph, the containing paragraph will be split.
InsertParagraphAfter(NamedStyle, ParagraphFormatting)	Creates and returns a Paragraph object that represents a new empty paragraph at the end of this Element. The paragraph that is inserted will have the style specified, to which any additional specified paragraph formatting is applied as well. If this Element is in the middle of a paragraph, the containing paragraph will be split.

Element.InsertParagraphAfter(NamedStyle, ParagraphFormatting)

Description

Creates and returns a Paragraph object that represents a new empty paragraph at the end of this Element. The paragraph that is inserted will have the style specified, to which any additional specified paragraph formatting is applied as well. If this Element is in the middle of a paragraph, the containing paragraph will be split.

Signature: C#

```
public virtual Paragraph InsertParagraphAfter(NamedStyle style, ParagraphFormatting props)
```

Signature: VB.NET

```
Public Overridable Function InsertParagraphAfter(ByVal style As NamedStyle, ByVal props As
ParagraphFormatting) As Paragraph
```

Parameters

style

A NamedStyle object representing the base style of this paragraph. The base style also affects the character style of any text added to this paragraph. If this is null, the style will be "Normal".

props

A ParagraphFormatting that represents formatting properties that should override the paragraph properties found in the style.

Returns

A `Paragraph` object that represents a new empty paragraph at the end of this Element. Even though the paragraph is empty there will always be at least a single carriage return.

Examples

C#

```
Paragraph p = e.InsertParagraphAfter(doc.Styles[NamedStyle.BuiltIn.Normal], pFormatting);
```

vb.net

```
Dim p As Paragraph = e.InsertParagraphAfter(doc.Styles(NamedStyle.BuiltIn.Normal), pFormatting)
```

Element.InsertParagraphAfter(NamedStyle)

Description

Creates and returns a Paragraph object that represents a new empty paragraph at the end of this Element. The paragraph that is inserted will have the style specified. If this Element is in the middle of a paragraph, the containing paragraph will be split.

Signature: C#

```
public virtual Paragraph InsertParagraphAfter(NamedStyle style)
```

Signature: VB.NET

```
Public Overridable Function InsertParagraphAfter(ByVal style As NamedStyle) As Paragraph
```

Parameters

style

A NamedStyle object representing the base style of this paragraph. The base style also affects the character style of any text added to this paragraph. If this is null, the style will be "Normal".

Returns

A `Paragraph` object that represents a new empty paragraph at the beginning of this Element. Even though the paragraph is empty there will always be at least a single carriage return.

Examples

C#

```
Paragraph p = e.InsertParagraphAfter(doc.Styles[NamedStyle.BuiltIn.Normal]);
```

vb.net

```
Dim p As Paragraph= e.InsertParagraphAfter(doc.Styles(NamedStyle.BuiltIn.Normal))
```

Element.InsertParagraphBefore

Overload	Description
InsertParagraphBefore(NamedStyle)	Creates and returns a Paragraph object that represents a new empty paragraph at the beginning of this Element. The paragraph that is inserted will have the style specified. If this Element is in the middle of a paragraph, the containing paragraph will be split.
InsertParagraphBefore(NamedStyle, ParagraphFormatting)	Creates and returns a Paragraph object that represents a new empty paragraph at the beginning of this Element. The paragraph that is inserted will have the style specified, to which any additional specified paragraph formatting is applied as well. If this Element is in the middle of a paragraph, the containing paragraph will be split.

Element.InsertParagraphBefore(NamedStyle, ParagraphFormatting)

Description

Creates and returns a Paragraph object that represents a new empty paragraph at the beginning of this Element. The paragraph that is inserted will have the style specified, to which any additional specified paragraph formatting is applied as well. If this Element is in the middle of a paragraph, the containing paragraph will be split.

Signature: C#

```
public virtual Paragraph InsertParagraphBefore(NamedStyle style, ParagraphFormatting props)
```

Signature: VB.NET

```
Public Overridable Function InsertParagraphBefore(ByVal style As NamedStyle, ByVal props As
ParagraphFormatting) As Paragraph
```

Parameters

style

A NamedStyle object representing the base style of this paragraph. The base style also affects the character style of any text added to this paragraph. If this is null, the style will be "Normal".

props

A ParagraphFormatting that represents formatting properties that should override the paragraph properties found in the style.

Returns

A `Paragraph` object that represents a new empty paragraph at the beginning of this Element. Even though the paragraph is empty there will always be at least a single carriage return.

Examples

C#

```
Paragraph p = e.InsertParagraphBefore(doc.Styles[NamedStyle.BuiltIn.Normal], pFormatting);
```

vb.net

```
Dim p As Paragraph = e.InsertParagraphBefore(doc.Styles(NamedStyle.BuiltIn.Normal), pFormatting)
```

Element.InsertParagraphBefore(NamedStyle)

Description

Creates and returns a Paragraph object that represents a new empty paragraph at the beginning of this Element. The paragraph that is inserted will have the style specified. If this Element is in the middle of a paragraph, the containing paragraph will be split.

Signature: C#

```
public virtual Paragraph InsertParagraphBefore(NamedStyle style)
```

Signature: VB.NET

```
Public Overridable Function InsertParagraphBefore(ByVal style As NamedStyle) As Paragraph
```

Parameters

style

A NamedStyle object representing the base style of this paragraph. The base style also affects the character style of any text added to this paragraph. If this is null, the style will be "Normal".

Returns

A `Paragraph` object that represents a new empty paragraph at the beginning of this Element. Even though the paragraph is empty there will always be at least a single carriage return.

Examples

C#

```
Paragraph p = e.InsertParagraphBefore(doc.Styles[NamedStyle.BuiltIn.Normal]);
```

vb.net

```
Dim p As Paragraph = e.InsertParagraphBefore(doc.Styles(NamedStyle.BuiltIn.Normal))
```

Element.InsertTableAfter

Overload	Description
InsertTableAfter(Int32, Int32)	Creates and returns a Table at the end of this Element. The table will contain the specified number of rows and columns.
InsertTableAfter(Int32, Int32, TableFormatting)	Creates and returns a Table at the end of this Element. The table will contain the specified number of rows and columns. It will also be formatted with the properties specified.

Element.InsertTableAfter(Int32, Int32, TableFormatting)

Description

Creates and returns a Table at the end of this Element. The table will contain the specified number of rows and columns. It will also be formatted with the properties specified.

Signature: C#

```
public virtual Table InsertTableAfter(int rows, int columns, TableFormatting props)
```

Signature: VB.NET

```
Public Overridable Function InsertTableAfter(ByVal rows As Integer, ByVal columns As Integer, ByVal
props As TableFormatting) As Table
```

Parameters

rows

An `int` representing the number of rows the table will have.

columns

An `int` representing the number of columns the table will have.

props

A TableFormatting object representing the formatting properties to apply to the table.

Returns

A `Table` object representing the newly created table.

Examples

C#

```
Table tbl = e.InsertTableAfter(2, 3, tFormatting);
```

vb.net

```
Dim tbl As Table = e.InsertTableAfter(2, 3, tFormatting)
```

Element.InsertTableAfter(Int32, Int32)

Description

Creates and returns a Table at the end of this Element. The table will contain the specified number of rows and columns.

Signature: C#

```
public virtual Table InsertTableAfter(int rows, int columns)
```

Signature: VB.NET

```
Public Overridable Function InsertTableAfter(ByVal rows As Integer, ByVal columns As Integer) As Table
```

Parameters

rows

An `int` representing the number of rows the table will have.

columns

An `int` representing the number of columns the table will have.

Returns

A `Table` object representing the newly created table.

Examples

C#

```
Table tbl = e.InsertTableAfter(2, 3);
```

vb.net

```
Dim tbl As Table = e.InsertTableAfter(2, 3)
```

Element.InsertTableBefore

Overload	Description
InsertTableBefore(Int32, Int32)	Creates and returns a Table at the beginning of this Element. The table will contain the specified number of rows and columns.
InsertTableBefore(Int32, Int32, TableFormatting)	Creates and returns a Table at the beginning of this Element. The table will contain the specified number of rows and columns. It will also be formatted with the properties specified.

Element.InsertTableBefore(Int32, Int32, TableFormatting)

Description

Creates and returns a Table at the beginning of this Element. The table will contain the specified number of rows and columns. It will also be formatted with the properties specified.

Signature: C#

```
public virtual Table InsertTableBefore(int rows, int columns, TableFormatting props)
```

Signature: VB.NET

```
Public Overridable Function InsertTableBefore(ByVal rows As Integer, ByVal columns As Integer, ByVal
props As TableFormatting) As Table
```

Parameters

rows

An `int` representing the number of rows the table will have.

columns

An `int` representing the number of columns the table will have.

props

A TableFormatting object representing the formatting properties to apply to the table.

Returns

A `Table` object representing the newly created table.

Examples

C#

```
Table tbl = e.InsertTableBefore(2, 3, tFormatting);
```

vb.net

```
Dim tbl As Table = e.InsertTableBefore(2, 3, tFormatting)
```

Element.InsertTableBefore(Int32, Int32)

Description

Creates and returns a Table at the beginning of this Element. The table will contain the specified number of rows and columns.

Signature: C#

```
public virtual Table InsertTableBefore(int rows, int columns)
```

Signature: VB.NET

```
Public Overridable Function InsertTableBefore(ByVal rows As Integer, ByVal columns As Integer) As
Table
```

Parameters

rows

An `int` representing the number of rows the table will have.

columns

An `int` representing the number of columns the table will have.

Returns

A `Table` object representing the newly created table.

Examples

C#

```
Table tbl = e.InsertTableBefore(2, 3);
```

vb.net

```
Dim tbl As Table = e.InsertTableBefore(2, 3)
```

Element.InsertTextAfter

Overload	Description
InsertTextAfter(String, Boolean)	Creates and returns a CharacterRun at the end of this Element. The text to be inserted is specified as well as whether this should be a new character run or part of the previous one.
InsertTextAfter(String, Font)	Creates and returns a CharacterRun at the end of this Element. The text to be inserted is specified as well as the font that should override the font found in the style of the character run's enclosing paragraph.

Element.InsertTextAfter(String, Boolean)

Description

Creates and returns a CharacterRun at the end of this Element. The text to be inserted is specified as well as whether this should be a new character run or part of the previous one.

Signature: C#

```
public virtual CharacterRun InsertTextAfter(System.String text, boolean newRun)
```

Signature: VB.NET

```
Public Overridable Function InsertTextAfter(ByVal text As String, ByVal newRun As Boolean) As
CharacterRun
```

Parameters

text

A `String` representing the text to insert.

newRun

A `boolean` representing if this should have default formatting and become its own character run (true) or if it should assume the formatting of the first character in this Element, becoming a part of that first character run.

Returns

The `CharacterRun` containing the inserted text.

Examples

C#

```
CharacterRun charRun = e.InsertTextAfter("Here is some text.", true);
```

vb.net

```
Dim charRun As CharacterRun = e.InsertTextAfter("Here is some text.", True)
```

Element.InsertTextAfter(String, Font)

Description

Creates and returns a CharacterRun at the end of this Element. The text to be inserted is specified as well as the font that should override the font found in the style of the character run's enclosing paragraph.

Signature: C#

```
public virtual CharacterRun InsertTextAfter(System.String text, Font props)
```

Signature: VB.NET

```
Public Overridable Function InsertTextAfter(ByVal text As String, ByVal props As Font) As CharacterRun
```

Parameters

text

A `String` representing the text to insert.

props

A Font object representing the formatting that should override the font found in the style of the character run's enclosing paragraph.

Returns

The `CharacterRun` containing the inserted text.

Examples

C#

```
CharacterRun charRun = e.InsertTextAfter("Here is some text.", fnt);
```

vb.net

```
Dim charRun As CharacterRun = e.InsertTextAfter("Here is some text.", fnt)
```

Element.InsertTextBefore

Overload	Description
InsertTextBefore(String, Boolean)	Creates and returns a CharacterRun at the beginning of this Element. The text to be inserted is specified as well as whether this should be a new character run or part of the existing one.
InsertTextBefore(String, Font)	Creates and returns a CharacterRun at the beginning of this Element. The text to be inserted is specified as well as the font that should override the font found in the style of the character run's enclosing paragraph.

Element.InsertTextBefore(String, Boolean)

Description

Creates and returns a CharacterRun at the beginning of this Element. The text to be inserted is specified as well as whether this should be a new character run or part of the existing one.

Signature: C#

```
public virtual CharacterRun InsertTextBefore(System.String text, boolean newRun)
```

Signature: VB.NET

```
Public Overridable Function InsertTextBefore(ByVal text As String, ByVal newRun As Boolean) As
CharacterRun
```

Parameters

text

A `String` representing the text to insert.

newRun

A `boolean` representing if this should have default formatting and become its own character run (true) or if it should assume the formatting of the first character in this Element, becoming a part of that first character run.

Returns

The `CharacterRun` containing the inserted text.

Examples

C#

```
CharacterRun charRun = e.InsertTextBefore("Here is some text.", true);
```

vb.net

```
Dim charRun As CharacterRun = e.InsertTextBefore("Here is some text.", True)
```

Element.InsertTextBefore(String, Font)

Description

Creates and returns a CharacterRun at the beginning of this Element. The text to be inserted is specified as well as the font that should override the font found in the style of the character run's enclosing paragraph.

Signature: C#

```
public virtual CharacterRun InsertTextBefore(System.String text, Font props)
```

Signature: VB.NET

```
Public Overridable Function InsertTextBefore(ByVal text As String, ByVal props As Font) As
CharacterRun
```

Parameters

text

A `String` representing the text to insert.

props

A Font object representing the font that should override the font found in the style of the character run's enclosing paragraph.

Returns

The `CharacterRun` containing the inserted text.

Examples

C#

```
CharacterRun charRun = e.InsertTextBefore("Here is some text.", fnt);
```

vb.net

```
Dim charRun As CharacterRun = e.InsertTextBefore("Here is some text.", fnt)
```

Element.Length

Description

Returns an `int` that represents the number of characters in this Element.

Signature: C#

```
public int Length{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Length() As Integer
```

Examples

C#

```
int length = e.Length;
```

vb.net

```
Dim length As Integer = e.Length
```

Element.Search(String)

Description

Searches for a specified string and returns a collection of type SearchMatch[].

Signature: C#

```
public virtual SearchMatch[] Search(System.String toMatch)
```

Signature: VB.NET

```
Public Overridable Function Search(ByVal toMatch As String) As SearchMatch()
```

Parameters

toMatch

The string to find.

Returns

A collection of SearchMatch objects.

Remarks

The search string can be a literal, a regular expression, or a combination of both. Since the string can contain regular expressions, use a backslash to escape special characters. Special characters are `[^$.|?*+(`. The collection this method returns implements the .NET IEnumerable interface that can be used to iterate over the search results.

Examples

C#

```
//--- Required for IEnumerator
          using System.Collections;

          WordApplication app = new WordAppliation();
          Document doc = app.Open(@"c:\myDoc.doc");

          //--- Search a document for social security numbers, or any series of
          //--- nine digits, and remove them. The question mark after each dash
          //--- makes the presence of the dash optional.
          IEnumerator searcherator = (doc.Search(@"\d\d\d-?\d\d-?\d\d\d\d")).GetEnumerator();
          while (searcherator.MoveNext())
          {
               SearchMatch match = (SearchMatch)searcherator.Current;
               match.Element.DeleteElement();
          }
```

vb.net

```
'--- Required for IEnumerator
          Imports System.Collections

          Dim app As New WordAppliation()
          Dim doc As Document = app.Open("c:\myDoc.doc")

          '--- Search a document for social security numbers, or any series of
          '--- nine digits, and remove them. The question mark after each dash
          '--- makes the presence of the dash optional.
          Dim searcherator As IEnumerator = (doc.Search("\d\d\d-?\d\d-?\d\d\d\d")).GetEnumerator()
          While searcherator.MoveNext()
               Dim match As SearchMatch = searcherator.Current
               match.Element.DeleteElement()
          End While
```

Element.SearchAndReplace(String, String)

Description

Searches for a string specified by the `search` parameter, and replaces it with the string specified by the `replaceWith` parameter. The search string can be a literal, a regular expression, or a combination of both. Since the string can contain regular expressions, use a backslash to escape special characters. Special characters are `[^$.|?*+()`.

Signature: C#

```
public virtual int SearchAndReplace(System.String search, System.String replaceWith)
```

Signature: VB.NET

```
Public Overridable Function SearchAndReplace(ByVal search As String, ByVal replaceWith As String) As
Integer
```

Parameters

search

The string to find and replace.

replaceWith

The replacement string.

Returns

The number of occurences that were found and replaced.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\myDoc.doc");

          //--- Update a copyright
          //--- Search for: (c) or the copyright symbol in ASCII, one or more whitespace characters,
          //--- four digits, and then any number of characters followed by 'Inc.'
          //--- Replace matches with '(c) 2008 SoftArtisans, Inc.'
          doc.SearchAndReplace("(\(c\)|\xA9)\s+[0-9]{4}\s+SoftArtisans, Inc\.", "(c) 2008
SoftArtisans, Inc.");
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\myDoc.doc")

          '--- Update a copyright
          '--- Search for: (c) or the copyright symbol in ASCII, one ore more whitespace characters,
          '--- four digits, and then any number of characters followed by 'Inc.'
          '--- Replace matches with '(c) 2008 SoftArtisans, Inc.'
          doc.SearchAndReplace("(\(c\)|\xA9)\s+[0-9]{4}\s+SoftArtisans, Inc\.", "(c) 2008
SoftArtisans, Inc.")
```

Element.Start

Description

Returns an `int` that represents the absolute character index of the start of this Element.

Signature: C#

```
public int Start{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Start() As Integer
```

Examples

C#

```
int start = e.Start;
```

vb.net

```
Dim start As Integer = e.Start
```

Element.SubElement(Int32, Int32)

Description

Creates and returns a Element object that represents a sub Element within this Element based on the specified start and end offsets.

Signature: C#

```
public virtual Element SubElement(int start, int end)
```

Signature: VB.NET

```
Public Overridable Function SubElement(ByVal start As Integer, ByVal [end] As Integer) As Element
```

Parameters

start

An `int` representing the starting offset of the sub Element relative to the start of this Element.

end

An `int` representing the ending offset of the sub Element relative the start of this Element.

Returns

A `Element` object that represents a sub Element within this Element.

Examples

C#

```
Element subE = e.SubElement();
```

vb.net

```
Dim subE As Element = e.SubElement()
```

Element.Text

Description

Returns a `String` that represents the text for this Element.

Signature: C#

```
public System.String Text{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Text() As String
```

Examples

C#

```
string text = e.Text;
```

vb.net

```
Dim text As String = e.Text
```

Element.Type

Description

An Element represents a region of the document that can be edited. An Element.Type value specifies the type of region an `Element` object represents.

Signature: C#

```
public sealed class Type : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Type
  Inherits TypesafeEnum
```

Values

Name	Description
Bookmark	A bookmark
CharacterRun	A contiguous run of characters in the document, all of which have the same formatting
Document	The whole Word document
Field	A generic representation of a field that encompasses many objects in Word, including merge fields, hyperlinks, and table of content entries.
HeaderFooter	A document's header and footer
Hyperlink	A hyperlink
InlineImage	An inline image (Word supports jpg, png, bmp, and gif images)
List	A numbered or bulleted list
ListEntry	A list item
MergeField	A merge field (a placeholder for data imported from a database)
Paragraph	A paragraph in the Word document (a block of text that ends with a carriage return)
Section	A major section in a document, like a chapter in a book

Table	A table
TableCell	A single cell in a table
Unspecified	Unspecified

Field

Description

This class is a low level representation of a field. This encompasses many objects in Word, such as merge fields, hyperlinks, TOC entries, etc. Fields are the generic form of what are contained in a Element's inline elements.

Signature: C#

```
public class Field : Element
```

Signature: VB.NET

```
Public Class Field
  Inherits Element
```

Remarks

To create a field, call the corresponding "insert" method for what you want to create (hyperlink, TOC entry, etc) on an Element or Position object.

To get an existing field, use the Element.GetElements() method and pass in the Element.Type.Field parameter to retrieve all fields.

The following sample demonstrates creating a field at the end of a new document and retrieving the first field of an existing document.

Examples

C#

```
//--- Insert a field at the end of a new document
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          Field fld =
               doc.InsertHyperlinkAfter("http://www.softartisans.com", "SoftArtisans");

          //--- Get the first field of an existing document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Field firstField = doc.GetElements(Element.Type.Field)[0];
```

vb.net

```
'--- Insert a field at the end of a new document
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim fld As Field = _
               doc.InsertHyperlinkAfter("http://www.softartisans.com", "SoftArtisans")

          '--- Get the first field of an existing document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim firstField As Field = doc.GetElements(Element.Type.Field)(0)
```

Properties

Name	Description

Code	Returns a `String` that represents this field's code.
Contents	Returns an Element object that represents the contents of this field.
Type	Returns a Field.FieldType object that represents this field's type.

Methods

Name	Description
Update()	Updates the value of the field.

Nested Classes

Name	Description
FieldType	The Field class is a general representation that encompasses many Word objects, including merge fields, hyperlinks, and table of contents items. A Field.FieldType value specifies the type of field a specific Field object represents.

Field.Code

Description

Returns a `String` that represents this field's code.

Signature: C#

```
public System.String Code{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Code() As String
```

Examples

C#

```
string fieldCode = fld.Code;
```

vb.net

```
Dim fieldCode As String = fld.Code
```

Field.Contents

Description

Returns an Element object that represents the contents of this field.

Signature: C#

```
public Element Contents{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Contents() As Element
```

Field.FieldType

Description

The Field class is a general representation that encompasses many Word objects, including merge fields, hyperlinks, and table of contents items. A Field.FieldType value specifies the type of field a specific Field object represents.

Signature: C#

```
public sealed class FieldType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class FieldType
  Inherits TypesafeEnum
```

Values

Name	Description
AddressBlock	A Mail Merge address block
Advance	Advance
AskCommand	Ask command (for Print Merge)
AutoNumArabic	Insert auto numbering field in Arabic number format
AutoNumLegal	Insert auto numbering field in legal format
AutoNumOutline	Insert auto numbering field in outline format
AutoText	AutoText
AutoTextList	AutoText list
BarCode	Bar code
BidiOutline	Set outline to RTL
BookmarkReference	Bookmark reference
Compare	Compare two values
CreateIndex	Creates an index, if index entries are defined
CreateTableOfAuthorities	Create table of authorities
CreateToc	Creates a table of contents
DataCommand	Data command (for Print Merge)
DDEAutoRef	DDE automatic reference
DDEReference	DDE reference
DocumentProp	Document property
DocumentReference	Document reference
DocumentVar	Document variable
EmbeddedObject	Embedded object
EvalExpression	Evaluate expression
FileSize	The document's file size
FillinCommand	Fill-in command to display prompt (for Print Merge)
FootnoteReference	Footnote
FormCheckBox	Form check box
FormListBox	Form list box

FormTxtBox	Form text box
Formula	Formula definition
GlossaryEntry	Glossary entry
GotoButton	Goto button
GreetingLine	A Mail Merge greeting line
HtmlControl	HTML control
Hyperlink	Hyperlink
IfCommand	If command (for Print Merge)
IncludeCommand	Include command (for Print Merge)
IncludePicture	Include picture
IncludeText	Include text
IndexEntry	An index entry
InsertDatabase	Insert database
InsertLiteralText	Insert literal text
Link	Link
ListElement	List element
Macro	Macro
MacroButton	Macro button
MarkTableOfAuthorities	Mark table of authorities
MergeField	Merge field
MergeRecSeqNum	Merge record sequence number
NextCommand	Next command (for Print Merge)
NextIfCommand	NextIf command (for Print Merge)
NoPrintTranslation	Sends characters to printer without translation
NoteRef	Note reference
NumPrintMerge	Inserts number of current Print Merge record
Ocx	OCX
PageReference	The number of the current page
PluginPrivate	Plug-in module private
PossibleBookmark	The field syntax matches bookmark name
Private	Private
PseudoInlineShape	Pseudo inline shape
QuoteAuthor	The name of the document's author as it appears in the Properties dialog's Summary sheet
QuoteComments	The text entered in the comments field of the document's Summary sheet
QuoteCreationDate	Date the document was created, as it appears in the Properties dialog's Statistics sheet
QuoteCurrentDate	The current date
QuoteCurrentPage	The current page in the document
QuoteCurrentTime	The current time
QuoteEditTime	The document's total editing time, as it appears in the Properties dialog's Statistics sheet

QuoteFileName	The document's file name
QuoteInfo	Data from the document's Summary sheet
QuoteKeywords	The keywords listed in the Properties dialog's Summary sheet
QuoteLastRevisedBy	Name of the person who last revised the document, as it appears in the Properties dialog's Statistics sheet
QuoteNumChars	The number of characters in the document
QuoteNumPages	The number of pages in the document
QuoteNumWords	The number of words in the document
QuotePrintDate	Date the document was last printed, as it appears in the Properties dialog's Statistics sheet
QuoteRevisionDate	Date the document was last revised, as it appears in the Properties dialog's Statistics sheet
QuoteRevisionNum	The number of times the document was revised, as it appears in the Properties dialog's Statistics sheet
QuoteSubject	The document's subject as it appears in the Properties dialog's Summary sheet
QuoteTemplateName	The document's template (Normal.dot by default)
QuoteTitle	The document's title as it appears in the Properties dialog's Summary sheet
Section	Section
SectionPages	Section pages
SequenceMark	Sequence mark
SetCommand	Set command (for Print Merge)
SkipIfCommand	SkipIf command (for Print Merge)
StyleReference	Style reference field
Subscriber	Subscriber
Symbol	Symbol
Tiff	Reads a TIFF file
TocEntry	Table of contents entry
Unknown	Unknown
UnknownKeyword	Unknown keyword
UserAddress	User address
UserInitial	User initial
UserName	User name

Field.Type

Description

Returns a Field.FieldType object that represents this field's type.

Signature: C#

```
public FieldType Type{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Type() As FieldType
```

Examples

C#

```
Field.FieldType fldType = fld.Type;
```

vb.net

```
Dim fldType As Field.FieldType = fld.Type
```

Field.Update()

Description

Updates the value of the field.

Signature: C#

```
public virtual void Update()
```

Signature: VB.NET

```
Public Overridable Sub Update()
```

Examples

C#

```
field.Update();
```

vb.net

```
field.Update()
```

Font

Description

The Font class is used to specify formatting that should be applied to a character run. It also represents the fonts of named styles in the document.

Signature: C#

```
public sealed class Font
```

Signature: VB.NET

```
Public NotInheritable Class Font
```

Remarks

There are two ways to obtain an instance of this class. The Document.CreateFont() method creates a new `Font` object based on the Normal style. Alternately, using NamedStyle.Font, you can create a `Font` object that is a copy of a particular style's font. The `Font` object that is returned can then be used in conjunction with methods in the Element class to create character runs, paragraphs, etc, with specific fonts.

The following example demonstrates both ways of getting a font, first by retrieving the Normal style's font from the document, second by retrieving a copy of the Heading1 style's font.

Examples

C#

```
//--- Get Normal font from Document
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          Font normalFont = doc.CreateFont();

          //--- Get font from Heading1 Style
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          Font heading1Font = doc.Styles[NamedStyle.BuiltIn.Heading1].Font;
```

vb.net

```
'--- Get Normal font from Document
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim normalFont As Font = doc.CreateFont()

          '--- Get font from Heading1 Style
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim heading1Font As Font = doc.Styles(NamedStyle.BuiltIn.Heading1).Font
```

Properties

Name	Description
Animation	Sets or returns a Font.TextAnimation object that represents the text animation for a character run.
Bold	Sets or returns a `boolean` that represents if a character run is bolded.
Border	Returns a Border object on which you can manipulate the border properties of a character run.
Caps	Sets or returns a `boolean` that represents if a character run has the All caps effect, formatting lowercase text as capital letters and reducing their size. All caps formatting does not affect numbers, punctutation, non-alphabetic chacters, or uppercase letters.
CharacterScaling	Sets or returns an `int` that represents the character scaling, stretching or compressing text horizontally as a percentage of its current size. CharacterScaling can be set to a value between 1 and 600 percent. Values beyond these will be automatically adjusted.
CharacterSpacing	Sets or returns an `int` that represents the spacing between characters in a character run in twips. One twip = (1/20 pt) or (1/1440 in). This number can be negative (decreased spacing) or positive (increased spacing).
DoubleStrike	Sets or returns a `boolean` that represents if a character run has the Double strikethrough effect, displaying a double line through the text.
Embossed	Sets or returns a `boolean` that represents if a character run has the Emboss effect, displaying the text as if it is raised off the page in relief.
Engraved	Sets or returns a `boolean` that represents if a character run has the Engrave effect, making the text to be imprinted or pressed into the page.
FontName	Sets or returns a `String` that represents the name of the font used in this formatting.
FontSize	Sets or returns an `float` that represents the font size for a character run. The font size will always be a multiple of 0.5.
Hidden	Sets or returns a `boolean` that represents if a character run has the Hidden effect set. If set to true, this font will only be displayed when the display hidden flag is turned on for the containing document.
Highlight	Sets or returns the highlight color for a character run as a `Color` object.
HyphenationRule	Sets or returns a Font.Hyphenation object that represents the hyphenation rule for a character run.
Italic	Sets or returns a `boolean` that represents if a character run is italicized.
Kerning	Sets or returns a `double` that represents the kerning threshold for a character run in points.

NoProofChecking	Sets or returns a `boolean` that represents if Word will not perform grammar and spell checking on a character run when the document is opened.
Outlined	Sets or returns a `boolean` that represents if a character run has the the Outline effect, displaying the inner and outer borders of each character. (This effect is not available for all fonts.)
Shading	Returns a Shading object which on which you can manipulate the shading (fill color and/or pattern) properties of a character run.
Shadow	Sets or returns a `boolean` that represents if a character run has the Shadow effect, displaying a shadow behind, beneath and to the right of each character.
SmallCaps	Sets or returns a `boolean` that represents if a character run has the Small caps effect, formatting lowercase text as capital letters and reducing their size. Small caps formatting does not affect numbers, punctutation, non-alphabetic chacters, or uppercase letters.
SuperSubScript	Sets or returns a ScriptType object that represents whether the text in this formatting is set to superscript, subscript, or none.
Strike	Sets or returns a `boolean` that represents if a character run has the Strikethrough effect, displaying a single line through the text.
TextColor	Sets or returns the color of the text in a character run as a `Color` object.
Underline	Sets or returns a Font.UnderlineType object that represents the underline type for this formatting.
UnderlineColor	Sets or returns the underline color for a character run as a `Color` object.
VerticalPosition	Sets or returns a `double` that represents the vertical position for a character run in half points, lowering or raising the text in relation to the baseline. (For example, 2 points would be 4 half points.)

Nested Classes

Name	Description
Hyphenation	Types of hyphenation rules that can be applied to a character run.
ScriptType	Values used to represent whether a character run is formatting as superscript, subscript, or neither.
TextAnimation	Types of text animation that can be set on a character run.
UnderlineType	Types of underlines that can be applied to a character run.

Font.Animation

Description

Sets or returns a Font.TextAnimation object that represents the text animation for a character run.

Signature: C#

```
public TextAnimation Animation{ get; set; }
```

Signature: VB.NET

```
Public Property Animation() As TextAnimation
```

Remarks

MS Word equivalent: Format menu > Font... > Text Effects tab > Animations:

Examples

C#

```
//--- Return TextAnimation
          Font.TextAnimation oTextAnimation = oFont.Animation;

          //--- Set TextAnimation to LasVegasLights
          oFont.Animation = Font.TextAnimation.LasVegasLights;
```

vb.net

```
'--- Return TextAnimation
          Dim oTextAnimation As Font.TextAnimation = oFont.Animation

          '--- Set TextAnimation to LasVegasLights
          oFont.Animation = Font.TextAnimation.LasVegasLights
```

Font.Bold

Description

Sets or returns a `boolean` that represents if a character run is bolded.

Signature: C#

```
public boolean Bold{ get; set; }
```

Signature: VB.NET

```
Public Property Bold() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Font section > Font style: > Bold (or Bold Italic if Italic = true)

Examples

C#

```
//--- Return Bold
          bool bold = oFont.Bold;

          //--- Set Bold
          oFont.Bold = true;
```

vb.net

```
'--- Return Bold
          Dim bold As Boolean = oFont.Bold

          '--- Set Bold
          oFont.Bold = True
```

Font.Border

Description

Returns a Border object on which you can manipulate the border properties of a character run.

Signature: C#

```
public Border Border{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Border() As Border
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Borders tab

Examples

C#

```
oFont.Border.LineStyle = Border.LineStyle.Double;
```

vb.net

```
oFont.Border.LineStyle = Border.LineStyle.Double
```

Font.Caps

Description

Sets or returns a `boolean` that represents if a character run has the All caps effect, formatting lowercase text as capital letters and reducing their size. All caps formatting does not affect numbers, punctutation, non-alphabetic chacters, or uppercase letters.

Signature: C#

```
public boolean Caps{ get; set; }
```

Signature: VB.NET

```
Public Property Caps() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > All caps

Examples

C#

```
//--- Return Caps
          bool caps = oFont.Caps;

          //--- Set Caps
          oFont.Caps = true;
```

vb.net

```
'--- Return Caps
          Dim caps As Boolean = oFont.Caps

          '--- Set Caps
          oFont.Caps = True
```

Font.CharacterScaling

Description

Sets or returns an `int` that represents the character scaling, stretching or compressing text horizontally as a percentage of its current size. CharacterScaling can be set to a value between 1 and 600 percent. Values beyond these will be automatically adjusted.

Signature: C#

```
public int CharacterScaling{ get; set; }
```

Signature: VB.NET

```
Public Property CharacterScaling() As Integer
```

Remarks

MS Word equivalent: Format menu > Font... > Character spacing tab > Scale:

Examples

C#

```
//--- Return CharacterScaling
          int charScale = oFont.CharacterScaling;

          //--- Set CharacterScaling
          oFont.CharacterScaling = 100;
```

vb.net

```
'--- Return CharacterScaling
          Dim charScale As Integer = oFont.CharacterScaling

          '--- Set CharacterScaling
          oFont.CharacterScaling = 100
```

Font.CharacterSpacing

Description

Sets or returns an `int` that represents the spacing between characters in a character run in twips. One twip = (1/20 pt) or (1/1440 in). This number can be negative (decreased spacing) or positive (increased spacing).

Signature: C#

```
public int CharacterSpacing{ get; set; }
```

Signature: VB.NET

```
Public Property CharacterSpacing() As Integer
```

Remarks

The minimum value is -1584 pts and the maximum value is 1584 pts. Values beyond this will be automatically adjusted.

MS Word equivalent: Format menu > Font... > Character spacing tab > Spacing:

Examples

C#

```
//--- Return CharacterSpacing
          int charSpacing = oFont.CharacterSpacing;

          //--- Set CharacterSpacing to 2 points
          oFont.CharacterSpacing = 40;
```

vb.net

```
'--- Return CharacterSpacing
          Dim charSpacing As Integer = oFont.CharacterSpacing

          '--- Set CharacterSpacing to 2 points
          oFont.CharacterSpacing = 40
```

Font.DoubleStrike

Description

Sets or returns a `boolean` that represents if a character run has the Double strikethrough effect, displaying a double line through the text.

Signature: C#

```
public boolean DoubleStrike{ get; set; }
```

Signature: VB.NET

```
Public Property DoubleStrike() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Double strikethrough

Examples

C#

```
//--- Return DoubleStrike
          bool doubleStrike = oFont.DoubleStrike;

          //--- Set Engraved
          oFont.DoubleStrike = true;
```

vb.net

```
'--- Return Engraved
          Dim doubleStrike As Boolean = oFont.DoubleStrike

          '--- Set Engraved
          oFont.DoubleStrike = True
```

Font.Embossed

Description

Sets or returns a `boolean` that represents if a character run has the Emboss effect, displaying the text as if it is raised off the page in relief.

Signature: C#

```
public boolean Embossed{ get; set; }
```

Signature: VB.NET

```
Public Property Embossed() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Embossed

Examples

C#

```
//--- Return Embossed
          bool embossed = oFont.Embossed;

          //--- Set Embossed
          oFont.Embossed = true;
```

vb.net

```
'--- Return Embossed
          Dim embossed As Boolean = oFont.Embossed

          '--- Set Embossed
          oFont.Embossed = True
```

Font.Engraved

Description

Sets or returns a `boolean` that represents if a character run has the Engrave effect, making the text to be imprinted or pressed into the page.

Signature: C#

```
public boolean Engraved{ get; set; }
```

Signature: VB.NET

```
Public Property Engraved() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Engrave

Examples

C#

```
//--- Return Engraved
          bool engraved = oFont.Engraved;

          //--- Set Engraved
          oFont.Engraved = true;
```

vb.net

```
'--- Return Engraved
          Dim engraved As Boolean = oFont.Engraved

          '--- Set Engraved
          oFont.Engraved = True
```

Font.FontName

Description

Sets or returns a `String` that represents the name of the font used in this formatting.

Signature: C#

```
public System.String FontName{ get; set; }
```

Signature: VB.NET

```
Public Property FontName() As String
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Font:

Examples

C#

```
//--- Return FontName
          string fontName = oFont.FontName;

          //--- Set Fontname
          oFont.FontName = "Arial";
```

vb.net

```
'--- Return FontName
          Dim fontName As String = oFont.FontName

          '--- Set FontName
          oFont.FontName = "Arial"
```

Font.FontSize

Description

Sets or returns an `float` that represents the font size for a character run. The font size will always be a multiple of 0.5.

Signature: C#

```
public double FontSize{ get; set; }
```

Signature: VB.NET

```
Public Property FontSize() As Double
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Size:

Examples

C#

```
//--- Return FontSize
          float fontSize = oFont.FontSize;

          //--- Set FontSize to 16 points
          oFont.FontSize = 16.0;
```

vb.net

```
'--- Return FontSize
          Dim fontSize As Float = oFont.FontSize

          '--- Set FontSize to 16 points
          oFont.FontSize = 16.0
```

Font.Hidden

Description

Sets or returns a `boolean` that represents if a character run has the Hidden effect set. If set to true, this font will only be displayed when the display hidden flag is turned on for the containing document.

Signature: C#

```
public boolean Hidden{ get; set; }
```

Signature: VB.NET

```
Public Property Hidden() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Hidden

Examples

C#

```
//--- Return Hidden
          bool hidden = oFont.Hidden;

          //--- Set Hidden
          oFont.Hidden = true;
```

vb.net

```
'--- Return Hidden
          Dim hidden As Boolean = oFont.Hidden

          '--- Set Hidden
          oFont.Hidden = True
```

Font.Highlight

Description

Sets or returns the highlight color for a character run as a `Color` object.

Signature: C#

```
public System.Drawing.Color Highlight{ get; set; }
```

Signature: VB.NET

```
Public Property Highlight() As System.Drawing.Color
```

Remarks

MS Word equivalent: Formatting toolbar > Highlight button

Examples

C#

```
//--- Return Highlight
          Color oColor = oFont.Highlight;

          //--- Set Highlight
          oFont.Highlight = Color.Yellow;
```

vb.net

```
'--- Return Highlight
          Dim oColor As Color = oFont.Highlight

          '--- Set Highlight
          oFont.Highlight = Color.Yellow
```

Font.Hyphenation

Description

Types of hyphenation rules that can be applied to a character run.

Signature: C#

```
public sealed class Hyphenation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Hyphenation
  Inherits TypesafeEnum
```

Remarks

Some hyphenation rules are deprecated and will have no effect.

Values

Name	Description
AddBefore	Deprecated

ChangeAfter	Deprecated
ChangeBefore	Deprecated
DeleteBefore	Deprecated
DeleteChangeBefore	Deprecated
None	No automatic hyphenation.
Normal	In a character run, a word that contains a hyphen is split at the hyphen and continued on the next line if it does not fit entirely on the current line.

Font.HyphenationRule

Description

Sets or returns a Font.Hyphenation object that represents the hyphenation rule for a character run.

Signature: C#

```
public Hyphenation HyphenationRule{ get; set; }
```

Signature: VB.NET

```
Public Property HyphenationRule() As Hyphenation
```

Remarks

MS Word equivalent: None

Examples

C#

```
//--- Return HyphenationRule
          Font.Hyphenation oHyphenationRule = oFont.HyphenationRule;

          //--- Set HyphenationRule
          oFont.HyphenationRule = Font.Hyphenation.ChangeAfter;
```

vb.net

```
'--- Return HyphenationRule
          Dim oHyphenationRule As Font.Hyphenation = oFont.HyphenationRule

          '--- Set HyphenationRule
          oFont.HyphenationRule = Font.Hyphenation.ChangeAfter
```

Font.Italic

Description

Sets or returns a `boolean` that represents if a character run is italicized.

Signature: C#

```
public boolean Italic{ get; set; }
```

Signature: VB.NET

```
Public Property Italic() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Font section > Font style: > Italic (or Bold Italic if Bold = true)

Examples

C#

```
//--- Return Italic
          bool italic = oFont.Italic;

          //--- Set Italic
          oFont.Italic = true;
```

vb.net

```
'--- Return Italic
          Dim italic As Boolean = oFont.Italic

          '--- Set Italic
          oFont.Italic = True
```

Font.Kerning

Description

Sets or returns a `double` that represents the kerning threshold for a character run in points.

Signature: C#

```
public double Kerning{ get; set; }
```

Signature: VB.NET

```
Public Property Kerning() As Double
```

Remarks

If this is zero, then kerning is disabled for a character run. The maximum value is 1638 points. Values beyond these will be automatically adjusted.

Examples

C#

```
//--- Return Kerning
          int kerning = oFont.Kerning;

          //--- Set Kerning to 12 pt
          oFont.Kerning = 24;
```

vb.net

```
'--- Return Kerning
          Dim kerning As Integer = oFont.Kerning

          '--- Set Kerning to 12 pt
          oFont.Kerning = 24
```

Font.NoProofChecking

Description

Sets or returns a `boolean` that represents if Word will not perform grammar and spell checking on a character run when the document is opened.

Signature: C#

```
public boolean NoProofChecking{ get; set; }
```

Signature: VB.NET

```
Public Property NoProofChecking() As Boolean
```

Remarks

MS Word equivalent: None

Examples

C#

```
//--- Return NoProofChecking
          bool noProofChecking = oFont.NoProofChecking;

          //--- Set NoProofChecking to not perform grammar and spell check
          oFont.NoProofChecking = true;
```

vb.net

```
'--- Return NoProofChecking
          Dim noProofChecking As Boolean = oFont.NoProofChecking

          '--- Set NoProofChecking to not perform grammar and spell check
          oFont.NoProofChecking = True
```

Font.Outlined

Description

Sets or returns a `boolean` that represents if a character run has the the Outline effect, displaying the inner and outer borders of each character. (This effect is not available for all fonts.)

Signature: C#

```
public boolean Outlined{ get; set; }
```

Signature: VB.NET

```
Public Property Outlined() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Outline

Examples

C#

```
//--- Return Outlined
          bool outlined = oFont.Outlined;

          //--- Set Outlined
          oFont.Outlined = true;
```

vb.net

```
'--- Return Outlined
          Dim outlined As Boolean = oFont.Outlined

          '--- Set Outlined
          oFont.Outlined = True
```

Font.ScriptType

Description

Values used to represent whether a character run is formatting as superscript, subscript, or neither.

Signature: C#

```
public sealed class ScriptType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ScriptType
  Inherits TypesafeEnum
```

Values

Name	Description
None	Character run is at default line height.
Sub	Character run is in subscript.
Super	Character run is in superscript.

Font.Shading

Description

Returns a Shading object which on which you can manipulate the shading (fill color and/or pattern) properties of a character run.

Signature: C#

```
public Shading Shading{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Shading() As Shading
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Shading tab

Font.Shadow

Description

Sets or returns a `boolean` that represents if a character run has the Shadow effect, displaying a shadow behind, beneath and to the right of each character.

Signature: C#

```
public boolean Shadow{ get; set; }
```

Signature: VB.NET

```
Public Property Shadow() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Shadow

Examples

C#

```
//--- Return Shadow
          bool shadow = oFont.Shadow;

          //--- Set Shadow
          oFont.Shadow = true;
```

vb.net

```
'--- Return Shadow
          Dim shadow As Boolean = oFont.Shadow

          '--- Set Shadow
          oFont.Shadow = True
```

Font.SmallCaps

Description

Sets or returns a `boolean` that represents if a character run has the Small caps effect, formatting lowercase text as capital letters and reducing their size. Small caps formatting does not affect numbers, punctutation, non-alphabetic chacters, or uppercase letters.

Signature: C#

```
public boolean SmallCaps{ get; set; }
```

Signature: VB.NET

```
Public Property SmallCaps() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Small caps

Examples

C#

```
//--- Return SmallCaps
          bool smallCaps = oFont.SmallCaps;

          //--- Set SmallCaps
          oFont.SmallCaps = true;
```

vb.net

```
'--- Return SmallCaps
          Dim smallCaps As Boolean = oFont.SmallCaps

          '--- Set SmallCaps
          oFont.SmallCaps = True
```

Font.Strike

Description

Sets or returns a `boolean` that represents if a character run has the Strikethrough effect, displaying a single line through the text.

Signature: C#

```
public boolean Strike{ get; set; }
```

Signature: VB.NET

```
Public Property Strike() As Boolean
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Strikethrough

Examples

C#

```
//--- Return Strike
          bool strike = oFont.Strike;

          //--- Set Shadow
          oFont.Strike = true;
```

vb.net

```
'--- Return Strike
          Dim strike As Boolean = oFont.Strike

          '--- Set Strike
          oFont.Strike = True
```

Font.SuperSubScript

Description

Sets or returns a ScriptType object that represents whether the text in this formatting is set to superscript, subscript, or none.

Signature: C#

```
public ScriptType SuperSubScript{ get; set; }
```

Signature: VB.NET

```
Public Property SuperSubScript() As ScriptType
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Effects section > Superscript and Subscript

Examples

C#

```
//--- Return ScriptType
          Font.ScriptType oSuperSubScript = oFont.SuperSubScript;

          //--- Set SuperSubScript to superscript
          oFont.SuperSubScript = Font.ScriptType.Super;
```

vb.net

```
'--- Return SuperSubScript
          Dim oSuperSubScript As Font.ScriptType = oFont.SuperSubScript

          '--- Set SuperSubScript to superscript
          oFont.SuperSubScript = Font.ScriptType.Super
```

Font.TextAnimation

Description

Types of text animation that can be set on a character run.

Signature: C#

```
public sealed class TextAnimation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class TextAnimation
  Inherits TypesafeEnum
```

Values

Name	Description
BackgroundBlink	The Word text animation named Blinking Background. The colors of both the text and background alternate between their originally assigned values and their opposite colors.
LasVegasLights	The Word text animation named Las Vegas Lights. The text is circumscribed by a rotating series of differently colored and differently shaped dots.
MarchingAnts	The Word text animation named Marching Black Ants. The text is circumscribed by a series of black dashes that move clockwise around the text.
None	No text animation.
RedMarchingAnts	The Word text animation named Marching Red Ants. The text is circumscribed by a series of red dashes that move clockwise around the text.

Shimmer	The Word text animation named Shimmer. The text alternates between its normal state and a slightly blurred one.
SparkleText	The Word text animation named Sparkle Text. A pattern of small, differently shaped dots of alternating colors appears in front of the text.

Font.TextColor

Description

Sets or returns the color of the text in a character run as a `Color` object.

Signature: C#

```
public System.Drawing.Color TextColor{ get; set; }
```

Signature: VB.NET

```
Public Property TextColor() As System.Drawing.Color
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Font color:

Examples

C#

```
//--- Return TextColor
          Color oColor = oFont.TextColor;

          //--- Set TextColor
          oFont.TextColor = Color.Red;
```

vb.net

```
'--- Return TextColor
          Dim oColor As Color = oFont.TextColor

          '--- Set TextColor
          oFont.TextColor = Color.Red
```

Font.Underline

Description

Sets or returns a Font.UnderlineType object that represents the underline type for this formatting.

Signature: C#

```
public UnderlineType Underline{ get; set; }
```

Signature: VB.NET

```
Public Property Underline() As UnderlineType
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Underline style:

Examples

C#

```
//--- Return Underline
          Font.UnderlineType oUnderline = oFont.Underline;

          //--- Set Underline to Double
          oFont.Underline = Font.UnderlineType.Double;
```

vb.net

```
'--- Return Underline
          Dim oUnderline As Font.UnderlineType = oFont.Underline

          '--- Set Underline to Double
          oFont.Underline = Font.UnderlineType.Double
```

Font.UnderlineColor

Description

Sets or returns the underline color for a character run as a `Color` object.

Signature: C#

```
public System.Drawing.Color UnderlineColor{ get; set; }
```

Signature: VB.NET

```
Public Property UnderlineColor() As System.Drawing.Color
```

Remarks

MS Word equivalent: Format menu > Font... > Font tab > Underline color:

Examples

C#

```
//--- Return UnderlineTypeColor
          Color oColor = oFont.UnderlineColor;

          //--- Set UnderlineColor
          oFont.UnderlineColor = Color.Blue;
```

vb.net

```
'--- Return UnderlineColor
          Dim oColor As Color = oFont.UnderlineColor

          '--- Set UnderlineColor
          oFont.UnderlineColor = Color.Blue
```

Font.UnderlineType

Description

Types of underlines that can be applied to a character run.

Signature: C#

```
public sealed class UnderlineType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class UnderlineType
  Inherits TypesafeEnum
```

Values

Name	Description
ByWord	Word underline type By Word. Sample Text
Dash	Word underline type Dash. Sample Text
DashedHeavy	Word underline type Dashed Heavy. Sample Text
DashLong	Word underline type Dash Long. Sample Text
DashLongHeavy	Word underline type Dash Long Heavy. Sample Text
Dot	Word underline type Dot. Sample Text
DotDash	Word underline type Dot Dash. Sample Text
DotDashHeavy	Word underline type Dot Dash Heavy. Sample Text
DotDashHeavy2	Word underline type Dot Dash Heavy 2. Sample Text
DotDotDash	Word underline type Dot Dot Dash. Sample Text
Dotted	Word underline type Dotted. Sample Text
DottedHeavy	Word underline type Dotted Heavy. Sample Text
Double	Word underline type Double. Sample Text
None	Word underline type none.
Single	Word underline type Single. Sample Text
Thick	Word underline type Thick. Sample Text
Wave	Word underline type Wave. Sample Text

WaveDouble	Word underline type Wave Double. Sample Text
WaveHeavy	Word underline type Wave Heavy. Sample Text

Font.VerticalPosition

Description

Sets or returns a `double` that represents the vertical position for a character run in half points, lowering or raising the text in relation to the baseline. (For example, 2 points would be 4 half points.)

Signature: C#

```
public double VerticalPosition{ get; set; }
```

Signature: VB.NET

```
Public Property VerticalPosition() As Double
```

Remarks

MS Word equivalent: Format menu > Font... > Character spacing tab > Position:

Examples

C#

```
//--- Return VerticalPosition
          int verticalPosition = oFont.VerticalPosition;

          //--- Set VerticalPosition to 2 points
          oFont.VerticalPosition = 4;
```

vb.net

```
'--- Return VerticalPosition
          Dim verticalPosition As Integer = oFont.VerticalPosition

          '--- Set VerticalPosition to 2 points
          oFont.VerticalPosition = 4
```

Hyperlink

Description

This class represents a hyperlink field in a document.

Signature: C#

```
public sealed class Hyperlink : Field
```

Signature: VB.NET

```
Public NotInheritable Class Hyperlink
  Inherits Field
```

Remarks

To create a hyperlink, use Element.InsertHyperlinkBefore(). To get the hyperlinks in a Document, use the Element.GetElements() method with a Element.Type.Hyperlink parameter.

The following sample demonstrates creating a hyperlink at the end of a new document and retrieving the first hyperlink of an existing document.

Examples

C#

```
//--- Add a hyperlink to a document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Hyperlink link =
               doc.InsertHyperlinkAfter("http://www.softartisans.com", "SoftArtisans");

          //--- Get the first hyperlink from a document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Hyperlink firstLink =
               doc.GetElements(Element.Type.Hyperlink)[0];
```

vb.net

```
'--- Add a hyperlink to a document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim link As Hyperlink = _
               doc.InsertHyperlinkAfter("http://www.softartisans.com", "SoftArtisans")

          '--- Get the first hyperlink from a document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim link As Hyperlink = _
               doc.GetElements(Element.Type.Hyperlink)(0)
```

Methods

Name	Description
GetUrlString()	Returns the URL that the hyperlink points to.

Hyperlink.GetUrlString()

Description

Returns the URL that the hyperlink points to.

Signature: C#

```
public System.String GetUrlString()
```

Signature: VB.NET

```
Public Function GetUrlString() As String
```

Examples

C#

```
string url = link.GetUrlString();
```

vb.net

```
Dim url As String = link.GetUrlString()
```

InlineImage

Description

This class is used to represent an inline image in a Word document. The types of images that Word supports are jpg, png, bmp, and gif. Inline images are part of the *InlineElement* objects of an Element.

Signature: C#

```
public sealed class InlineImage : Element
```

Signature: VB.NET

```
Public NotInheritable Class InlineImage
  Inherits Element
```

Remarks

To create a new inline image, use the insert methods for inline images from the `Element` or Position class.

Examples

The following sample demonstrates creating a inline image at the end of a new document and retrieving the first inline image of an existing document.

C#

```
//--- Insert a inline image at the end of a new document
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          InlineImage image = doc.InsertImageAfter(@"C:\sample.jpg");

          //--- Get the first inline image of an existing document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          InlineImage firstImage =
               (InlineImage)doc.GetElements(Element.Type.InlineImage)[0];
```

vb.net

```
'--- Insert an inline image at the end of a new document
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim image As InlineImage = doc.InsertImageAfter("C:\sample.jpg")

          '--- Get the first inline image of an existing document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim firstImage As InlineImage = _
               doc.GetElements(Element.Type.InlineImage)(0)
```

Properties

Name	Description

Height	Sets or returns an `int` that represents the image's height in twips. One twip = (1/20 pt) or (1/1440 in).
Image	Sets or returns the image in its default format.
ImageFormat	Returns an InlineImage.Format object that represents the format that the image is stored as in Word. (Jpg, Png, Bmp, Gif)
Width	Sets or returns an `int` that represents the image's width in twips. One twip = (1/20 pt) or (1/1440 in).

Methods

Name	Description
GetImage(InlineImage.Format)	Returns the image in the given format.

Nested Classes

Name	Description
Format	Image formats for an inline image.

InlineImage.Format

Description

Image formats for an inline image.

Signature: C#

```
public sealed class Format : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Format
  Inherits TypesafeEnum
```

Values

Name	Description
Bmp	Bitmap image format
Unknown	
Emf	Microsoft enhanced metafile format
Gif	Graphics interchange format
Jpg	Joint Photographic Experts Group (jpeg) format
Pict	PICT graphic file format
Png	Portable Network Graphics format
Wmf	Microsoft Windows metafile format

InlineImage.GetImage(InlineImage.Format)

Description

Returns the image in the given format.

Signature: C#

```
public System.IO.Stream GetImage(Format imgFormat)
```

Signature: VB.NET

```
Public Function GetImage(ByVal imgFormat As Format) As System.IO.Stream
```

Parameters

format

An InlineImage.Format of the type you wish to return.

Returns

A `System.IO.Stream` object that represents the image that was inserted.

Exceptions

System.Exception

Examples

C#

```
System.IO.Stream image = firstImage.GetImage(InlineImage.Bmp);
```

vb.net

```
Dim image As System.IO.Stream = firstImage.GetImage(InlineImage.Bmp)
```

InlineImage.Height

Description

Sets or returns an `int` that represents the image's height in twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Integer
```

Examples

C#

```
//--- Return Height
          int height = oInlineImage.Height;

          //--- Set Height to 2 inches
          oInlineImage.Height = 2880;
```

vb.net

```
'--- Return Height
          Dim height As Integer = oInlineImage.Height

          '--- Set Height to 2 inches
          oInlineImage.Height = 2880
```

InlineImage.Image

Description

Sets or returns the image in its default format.

Signature: C#

```
public System.IO.Stream Image{ get; set; }
```

Signature: VB.NET

```
Public Property Image() As System.IO.Stream
```

Examples

C#

```
//--- Return Image
          Stream oStream = oInlineImage.Image;

          //--- Set Image
          oInlineImage.Image = oStream;
```

vb.net

```
'--- Return Image
          Dim oStream As Stream = oInlineImage.Image

          '--- Set Image
          oInlineImage.Image = oStream
```

InlineImage.ImageFormat

Description

Returns an InlineImage.Format object that represents the format that the image is stored as in Word. (Jpg, Png, Bmp, Gif)

Signature: C#

```
public Format ImageFormat{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ImageFormat() As Format
```

Examples

C#

```
InlineImage.Format oImageFormat = oInlineImage.ImageFormat;
```

vb.net

```
Dim oImageFormat As InlineImage.Format = oInlineImage.ImageFormat
```

InlineImage.Width

Description

Sets or returns an `int` that represents the image's width in twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Integer
```

Examples

C#

```
//--- Return Width
          int width = oInlineImage.Width;

          //-- Set Width to 3 inches
          oInlineImage.Width = 4320;
```

vb.net

```
'--- Return Width
          Dim width As Integer = oInlineImage.Width

          '--- Set Width to 3 inches
          oInlineImage.Width = 4320
```

List

Description

This class is used to represent a list in a Word document. Lists are part of the *BlockElement* objects of an Element. Using this class, Lists can be read and written. Lists contain list entries, which are basically paragraphs with additional features.

Signature: C#

```
public sealed class List : Element
```

Signature: VB.NET

```
Public NotInheritable Class List
  Inherits Element
```

Remarks

To create a new list, use Element.InsertListBefore() or Element.InsertListAfter(). To get an existing list, use Element.GetElements(Element.Type).

Examples

The following sample demonstrates how to create a new list at the end of a document as well as how to get the first list of a document.

C#

```
//--- Insert a numbered list at the end of a new document with one entry
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          List lst = doc.InsertListAfter(true);
          lst.AddEntry(0);
          lst.InsertTextAfter("First list entry.", false);

          //--- Get the first list of an existing document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          List lst = doc.GetElements(Element.Type.List)[0];
```

vb.net

```
'--- Insert a numbered list at the end of a new document with one entry
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim lst As List = doc.InsertListAfter(True)
          lst.AddEntry(0)
          lst.InsertTextAfter("First list entry.", False)

          '--- Get the first list of an existing document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim lst As List = doc.GetElements(Element.Type.List)(0)
```

Properties

Name	Description
NumEntries	Returns an `int` representing the number of contiguous entries in this lst.

Methods

Name	Description
AddEntry(Int32)	Returns a ListEntry object representing an empty list entry added to the end of the list, which is indented to the level specified. Level can be between 0 and 8 (9 levels total).
GetEntry(Int32)	Returns a ListEntry object representing the entry at a specified index. The index of list entries starts at 0.
GetLevel(Int32)	Returns a ListLevel object representing a particular level for this lst. The levels start at 0 and end at 8 (9 levels total).
InsertEntry(Int32, Int32)	Returns a ListEntry object representing an empty list entry that is inserted at the given index, which is indented to the level specified. index and level start at 0. The maximum level is 8 (9 levels total).

List.AddEntry(Int32)

Description

Returns a ListEntry object representing an empty list entry added to the end of the list, which is indented to the level specified. Level can be between 0 and 8 (9 levels total).

Signature: C#

```
public ListEntry AddEntry(int level)
```

Signature: VB.NET

```
Public Function AddEntry(ByVal level As Integer) As ListEntry
```

Parameters

level

An `int` representing the level to assign to the new entry.

Returns

A `ListEntry` object representing the newly added entry.

Examples

C#

```
ListEntry entry = lst.AddEntry(0);
```

vb.net

```
Dim entry As ListEntry = lst.AddEntry(0)
```

List.GetEntry(Int32)

Description

Returns a ListEntry object representing the entry at a specified index. The index of list entries starts at 0.

Signature: C#

```
public ListEntry GetEntry(int index)
```

Signature: VB.NET

```
Public Function GetEntry(ByVal index As Integer) As ListEntry
```

Parameters

index

An `int` representing the level to get the list entry at.

Returns

A `ListEntry` object representing the entry at a specified index.

Examples

C#

```
ListEntry entry = lst.getEntry(0);
```

vb.net

```
Dim entry As ListEntry = lst.getEntry(0)
```

List.GetLevel(Int32)

Description

Returns a ListLevel object representing a particular level for this lst. The levels start at 0 and end at 8 (9 levels total).

Signature: C#

```
public ListLevel GetLevel(int level)
```

Signature: VB.NET

```
Public Function GetLevel(ByVal level As Integer) As ListLevel
```

Parameters

level

An `int` representing the level to get the list level at.

Returns

A `ListLevel` object representing a particular level for this lst.

Examples

C#

```
ListLevel level = lst.GetLevel(0);
```

vb.net

```
Dim level As ListLevel = lst.GetLevel(0)
```

List.InsertEntry(Int32, Int32)

Description

Returns a ListEntry object representing an empty list entry that is inserted at the given index, which is indented to the level specified. index and level start at 0. The maximum level is 8 (9 levels total).

Signature: C#

```
public ListEntry InsertEntry(int index, int level)
```

Signature: VB.NET

```
Public Function InsertEntry(ByVal index As Integer, ByVal level As Integer) As ListEntry
```

Parameters

index

An `int` representing the index at which to insert the new entry.

level

An `int` representing the level to assign to the new entry.

Returns

A `ListEntry` object representing the newly inserted entry.

Examples

C#

```
ListEntry entry = lst.InsertEntry(0, 0);
```

vb.net

```
Dim entry As ListEntry = lst.InsertEntry(0, 0)
```

List.NumEntries

Description

Returns an `int` representing the number of contiguous entries in this lst.

Signature: C#

```
public int NumEntries{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NumEntries() As Integer
```

Examples

C#

```
int numEntries = lst.NumEntries;
```

vb.net

```
Dim numEntries As Integer = lst.NumEntries
```

ListEntry

Description

This class is used to represent an entry in a List inside a Word document. A list entry in Word is basically a numbered paragraph.

Signature: C#

```
public sealed class ListEntry : Paragraph
```

Signature: VB.NET

```
Public NotInheritable Class ListEntry
  Inherits Paragraph
```

Remarks

The following sample gets the first ListEntry object of a List.

To create a new list entry, use List.AddEntry(). To get an existing list entry, use Element.GetElements(Element.Type.ListEntry) to get all elements of type `ListEntry`.

Examples

The following sample demonstrates how to create a new list entry at the end of a list as well as how to get the first list entry of an existing list.

C#

```
//--- Create a new list with a new list entry
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          List lst = doc.InsertListAfter(true);
          lst.AddEntry(0);
          lst.InsertTextAfter("First list entry.", false);

          //--- Get the first list entry of an existing list
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          ListEntry entry =
               (ListEntry)doc.GetElements(Element.Type.ListEntry)[0];
```

vb.net

```
'--- Create a new list with a new list entry
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim lst As List = doc.InsertListAfter(True)
          lst.AddEntry(0)
          lst.InsertTextAfter("First list entry.", False)

          '--- Get the first list entry of an existing list
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim entry As ListEntry = _
               doc.GetElements(Element.Type.ListEntry)(0)
```

Properties

Name	Description
BulletText	Sets or returns a `String` representing the bullet text of this entry as it would show up if it was opened in Word.
LevelNum	Sets or returns an `int` representing the indent level for this list entry.

Methods

Name	Description
RestartNumbering()	Sets the number of this list entry to "1" and renumbers all subsequent entries in the List. This method splits the current ListEntry and all subsequent list entries into a new `List`.

ListEntry.BulletText

Description

Sets or returns a `String` representing the bullet text of this entry as it would show up if it was opened in Word.

Signature: C#

```
public System.String BulletText{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property BulletText() As String
```

Examples

C#

```
string bulletText = entry.BulletText;
```

vb.net

```
Dim bulletText As String = entry.BulletText
```

ListEntry.LevelNum

Description

Sets or returns an `int` representing the indent level for this list entry.

Signature: C#

```
public int LevelNum{ get; set; }
```

Signature: VB.NET

```
Public Property LevelNum() As Integer
```

Examples

C#

```
//--- Return LevelNum
          int levelNum = entry.LevelNum;

          //--- Set LevelNum to 4th level
          entry.LevelNum = 3;
```

vb.net

```
'--- Return LevelNum
          Dim levelNum As Integer = entry.LevelNum

          '--- Set LevelNum to 4th level
          entry.LevelNum = 3
```

ListEntry.RestartNumbering()

Description

Sets the number of this list entry to "1" and renumbers all subsequent entries in the List. This method splits the current ListEntry and all subsequent list entries into a new `List`.

Signature: C#

```
public List RestartNumbering()
```

Signature: VB.NET

```
Public Function RestartNumbering() As List
```

ListLevel

Description

This class is used to manipulate settings for a particular indent level. Things such as number and bullet text and indent change with each indent level in Word.

Signature: C#

```
public sealed class ListLevel
```

Signature: VB.NET

```
Public NotInheritable Class ListLevel
```

Remarks

You cannot create a new ListLevel object using the `new` keyword. To get an instance of the `ListLevel` class, use the GetLevel() method of the List class, passing it the index of the level you want to retrieve.

To get an existing list entry, use the `getEntry()` method of the `List` class. `getEntry` accepts a parameter that represents the index of the entry to retrieve.

The following sample demonstrates how to get a `ListLevel` object for level 2 of a list.

Examples

C#

```
//--- Get the second level settings of a list
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          List lst = doc.Lists[0];
          ListEntry entry = lst.GetLevel(2);
```

vb.net

```
'--- Get the second level settings of a list
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim lst As List = doc.Lists(0)
          Dim level As ListLevel = lst.GetLevel(2)
```

Properties

Name	Description
BulletText	Sets the bullet text that will be displayed at the current list level. It uses a pattern. If the text is ASCII codes 0-8, this represents the position of the list entry number for a particular level. A list is at a maximum nine levels deep. For example, in an outline you will have numbering that changes with the level such as 1.1, 1.1.2, and so on.
Indent	Sets or returns an `int` that represents the indentation for the body of the entry, in twips.
NumberFormat	Sets or returns a WordNumberFormat object that represents the format for list numbers for this list level.
NumberIndent	Sets or returns an `int` that represents the indent for the first line of the entry in twips. One twip = (1/20 pt) or (1/1440 in).

ListLevel.BulletText

Description

Sets the bullet text that will be displayed at the current list level. It uses a pattern. If the text is ASCII codes 0-8, this represents the position of the list entry number for a particular level. A list is at a maximum nine levels deep. For example, in an outline you will have numbering that changes

with the level such as 1.1, 1.1.2, and so on.

Signature: C#

```
public System.String BulletText{ set; }
```

Signature: VB.NET

```
Public WriteOnly Property BulletText() As String
```

Examples

C#

```
//--- Set the 3rd list level to 1.2
          level.BulletText = "\u0000.\u0001.\u0002";
```

vb.net

```
'--- Set the 3rd list level to 1.2
          level.BulletText = "\u0000.\u0001.\u0002"
```

ListLevel.Indent

Description

Sets or returns an `int` that represents the indentation for the body of the entry, in twips.

Signature: C#

```
public int Indent{ get; set; }
```

Signature: VB.NET

```
Public Property Indent() As Integer
```

Remarks

One twip = 1/20 pt = 1/1440 in. Indent is equivalent to Microsoft Word's "Text position" setting. WordWriter's default `Indent` value for the first list level is 0. By default, WordWriter increments the indentation of lower levels by 0.5".

To change Word's "Text Position" setting:

1. Highlight your list.
2. Open the Format menu and select Bullets and Numbering.
3. Select a list style and click Customize.

Indent is a read / write property.

Examples

C#

```
//--- Return Indent
          int indent = level.Indent;

          //--- Set Indent to 1 inch
          level.Indent = 1440;
```

vb.net

```
'--- Return Indent
          Dim indent As Integer = level.Indent

          '--- Set Indent to 1 inch
          level.Indent = 1440
```

ListLevel.NumberFormat

Description

Sets or returns a WordNumberFormat object that represents the format for list numbers for this list level.

Signature: C#

```
public WordNumberFormat NumberFormat{ get; set; }
```

Signature: VB.NET

```
Public Property NumberFormat() As WordNumberFormat
```

Examples

C#

```
//--- Return NumberFormat
          WordNumberFormat numberFormat = level.NumberFormat;

          //--- Set NumberFormat
          level.NumberFormat = WordNumberFormat.Ordinal;
```

vb.net

```
'--- Return NumberFormat
          Dim numberFormat As WordNumberFormat = level.NumberFormat

          '--- Set NumberFormat
          level.NumberFormat = WordNumberFormat.Ordinal
```

ListLevel.NumberIndent

Description

Sets or returns an `int` that represents the indent for the first line of the entry in twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int NumberIndent{ get; set; }
```

Signature: VB.NET

```
Public Property NumberIndent() As Integer
```

Examples

C#

```
//--- Return NumberIndent
          int numberIndent = level.NumberIndent;

          //--- Set NumberIndent to 1/2 inch
          level.NumberIndent = 720;
```

vb.net

```
'--- Return NumberIndent
          Dim numberIndent As Integer = level.NumberIndent

          '--- Set NumberIndent to 1/2 inch
          level.NumberIndent = 720
```

MergeField

Description

This class represents a merge field in a document. Merge fields are used in WordWriter templates as data placeholders.

Signature: C#

```
public sealed class MergeField : Field
```

Signature: VB.NET

```
Public NotInheritable Class MergeField
  Inherits Field
```

Remarks

To create a merge field, use Element.InsertMergeFieldBefore or Element.InsertMergeFieldAfter.

The following sample demonstrates creating a merge field at the end of a new document and retrieving the first merge field of an existing document.

Examples

C#

```
//--- Insert a merge field at the end of a new document
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          MergeField mfield = doc.InsertMergeFieldAfter("FieldName", "FieldName");

          //--- Get the first merge field of an existing document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          MergeField firstMergeField = doc.GetElements(Element.Type.MergeField)[0];
```

vb.net

```
'--- Insert a merge field at the end of a new document
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim mfield As Field = doc.InsertMergeFieldAfter("FieldName", "FieldName")

          '--- Get the first merge field of an existing document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim firstMergeField As MergeField = doc.GetElements(Element.Type.MergeField)(0)
```

Methods

Name	Description
GetFieldName()	Returns the merge field's name.

MergeField.GetFieldName()

Description

Returns the merge field's name.

Signature: C#

```
public System.String GetFieldName()
```

Signature: VB.NET

```
Public Function GetFieldName() As String
```

Returns

The name assigned to the merge field.

NamedStyle

Description

The NamedStyle class represents a named style, such as "Normal", "Heading 1", etc., that is embedded in a Word document.

Signature: C#

```
public sealed class NamedStyle
```

Signature: VB.NET

```
Public NotInheritable Class NamedStyle
```

Remarks

You cannot create a new named style programmatically. You can only work with Named Styles that are in an existing document. You can obtain an instance of this class from the Document.Styles collection. You can apply a style to different sections and elements within your document. You cannot, however, modify a named style using WordWriter.

The following sample retrieves a copy of the Heading 1 style and applies it to a new paragraph.

Examples

C#

```
NamedStyle heading1Style = doc.Styles[NamedStyle.BuiltIn.Heading1];
          Paragraph p = doc.InsertParagraphAfter(heading1Style);
```

vb.net

```
Dim heading1Style As NamedStyle = doc.Styles(NamedStyle.BuiltIn.Heading1)
          Dim p As Paragraph = doc.InsertParagraphAfter(heading1Style)
```

Properties

Name	Description
BuiltInStyleName	Returns a NamedStyle.BuiltIn object that represents the culture-independent name of this style. This is only useful for built-in styles that come with Word. Any custom styles will return UserDefined.
Font	Returns a Font object that is a copy of this style's font. Some style types do not have character formatting. In these cases, this method returns null.
Name	Returns a `String` representing the name of this NamedStyle.
ParagraphFormatting	Returns a ParagraphFormatting object that is a copy of this style's paragraph formatting. Some style types do not have paragraph formatting. In these cases, this method returns null.
StyleType	Returns a NamedStyle.Type object that represents the type of this style. Word 2003 supports four style types: Character, Paragraph, Table, and List. Older versions support fewer than this.
TableFormatting	Returns a TableFormatting object that is a copy of this style's table formatting. Some style types do not have table formatting. In these cases, this method returns null.

Nested Classes

Name	Description
BuiltIn	Named styles that are built-in to Word.
Type	Types of named styles.

NamedStyle.BuiltIn

Description

Named styles that are built-in to Word.

Signature: C#

```
public sealed class BuiltIn : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BuiltIn
  Inherits TypesafeEnum
```

Values

Name	Description
Acetate	
AnnotationReference	
AnnotationSubject	

AnnotationText	
AutoSignature	
BlockQuote	
BodyText	
BodyText2	
BodyText3	
BodyTextFirstIndent	
BodyTextFirstIndent2	
BodyTextIndent2	
BodyTextIndent3	
BodyTextIndented	
Caption	
Closing	
Date	
Emphasis	
EndnoteReference	
EndnoteText	
EnvelopeAddress	
EvelopeReturn	
Footer	
FootnoteReference	
FootnoteText	
FormTop	
FormBottom	
Header	
Heading1	Font: Arial, 16 pt, bold, space before: 12 pt, space after: 3 pt
Heading2	Font: Arial 14 pt, bold, italic, space before 12 pt, space after 3 pt
Heading3	Font: Arial 13 pt, bold, space before 12 pt, space after 3 pt
Heading4	Font: Times New Roman, 14 pt, bold, space before 12 pt, space after 3 pt
Heading5	Font: Times New Roman, 13 pt, bold, italic, space before 12 pt, space after 3 pt
Heading6	Font: Times New Roman, 11 pt, bold, space before 12 pt, space after 3 pt
Heading7	Font: Times New Roman, 12 pt, space before 12 pt, space after 3 pt
Heading8	Font: Times New Roman, 12 pt, italic, space before 12 pt, space after 3 pt
Heading9	Font: Arial, 11 pt, space before 12 pt, space after 3 pt
HtmlAcronym	
HtmlAddress	
HtmlCite	
HtmlCode	
HtmlDefinition	

HtmlKeyboard	
HtmlNormal	
HtmlPreformatted	
HtmlSample	
HtmlTypewriter	
HtmlVariable	
Hyperlink	
HyperlinkFollowed	
Index1	Index level 1
Index2	Index level 2
Index3	Index level 3
Index4	Index level 4
Index5	Index level 5
Index6	Index level 6
Index7	Index level 7
Index8	Index level 8
Index9	Index level 9
IndexHeading	
LineNumber	
List	
List2	
List3	
List4	
List5	
ListBullet	
ListBullet2	
ListBullet3	
ListBullet4	
ListBullet5	
ListContinue	
ListContinue2	
ListContinue3	
ListContinue4	
ListContinue5	
ListNumber	
ListNumber2	
ListNumber3	
ListNumber4	
ListNumber5	

Macro	
MessageHeader	
NavPane	
Normal	Font: Times New Roman, 12 pt
NormalChar	
NormalIndent	Indent 0.5"
NormalList	
NormalTable	
NoteHeading	
OutlineList1	
OutlineList2	
OutlineList3	
PageNumber	
PlainText	
Salutation	
Signature	
Strong	
Subtitle	
Table3dFx1	
Table3dFx2	
Table3dFx3	
TableClassic1	
TableClassic2	
TableClassic3	
TableClassic4	
TableColorful1	
TableColorful2	
TableColorful3	
TableColumns1	
TableColumns2	
TableColumns3	
TableColumns4	
TableColumns5	
TableContemporary	
TableElegant	
TableGrid	
TableGrid1	
TableGrid2	
TableGrid3	

TableGrid4	
TableGrid5	
TableGrid6	
TableGrid7	
TableGrid8	
TableList1	
TableList2	
TableList3	
TableList4	
TableList5	
TableList6	
TableList7	
TableList8	
TableProfessional	
TableSimple1	
TableSimple2	
TableSimple3	
TableSubtle1	
TableSubtle2	
TableTheme	
TableWeb1	
TableWeb2	
TableWeb3	
Title	
TOA	
TOAHeading	
Toc1	Table of contents level 1
Toc2	Table of contents level 2
Toc3	Table of contents level 3
Toc4	Table of contents level 4
Toc5	Table of contents level 5
Toc6	Table of contents level 6
Toc7	Table of contents level 7
Toc8	Table of contents level 8
Toc9	Table of contents level 9
ToCaption	
UserDefined	

NamedStyle.BuiltInStyleName

Description

Returns a NamedStyle.BuiltIn object that represents the culture-independent name of this style. This is only useful for built-in styles that come with Word. Any custom styles will return UserDefined.

Signature: C#

```
public BuiltIn BuiltInStyleName{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property BuiltInStyleName() As BuiltIn
```

Examples

C#

```
NamedStyle.BuiltIn builtInStyle = namedStyl.BuiltInStyleName;
```

vb.net

```
Dim builtInStyle As NamedStyle.BuiltIn = namedStyl.BuiltInStyleName
```

NamedStyle.Font

Description

Returns a Font object that is a copy of this style's font. Some style types do not have character formatting. In these cases, this method returns null.

Signature: C#

```
public Font Font{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Font() As Font
```

Examples

C#

```
Font fnt = namedStyl.Font;
```

vb.net

```
Dim fnt As Font = namedStyl.Font
```

NamedStyle.Name

Description

Returns a `String` representing the name of this NamedStyle.

Signature: C#

```
public System.String Name{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Name() As String
```

Examples

C#

```
string name = namedStyl.Name;
```

vb.net

```
Dim name As String = namedStyl.Name
```

NamedStyle.ParagraphFormatting

Description

Returns a ParagraphFormatting object that is a copy of this style's paragraph formatting. Some style types do not have paragraph formatting. In these cases, this method returns null.

Signature: C#

```
public ParagraphFormatting ParagraphFormatting{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ParagraphFormatting() As ParagraphFormatting
```

Examples

C#

```
ParagraphFormatting pFormatting = namedStyl.ParagraphFormatting;
```

vb.net

```
Dim pFormatting As ParagraphFormatting = namedStyl.ParagraphFormatting
```

NamedStyle.StyleType

Description

Returns a NamedStyle.Type object that represents the type of this style. Word 2003 supports four style types: Character, Paragraph, Table, and List. Older versions support fewer than this.

Signature: C#

```
public Type StyleType{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property StyleType() As Type
```

Examples

C#

```
NamedStyle.Type nsType = namedStyl.Type;
```

vb.net

```
Dim nsType As NamedStyle.Type = namedStyl.Type
```

NamedStyle.TableFormatting

Description

Returns a TableFormatting object that is a copy of this style's table formatting. Some style types do not have table formatting. In these cases, this method returns null.

Signature: C#

```
public TableFormatting TableFormatting{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property TableFormatting() As TableFormatting
```

Examples

C#

```
TableFormatting tFormatting = namedStyl.TableFormatting;
```

vb.net

```
Dim tFormatting As TableFormatting = namedStyl.TableFormatting
```

NamedStyle.Type

Description

Types of named styles.

Signature: C#

```
public sealed class Type : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Type
  Inherits TypesafeEnum
```

Values

Name	Description
Character	Character style can contain character formatting only, including font, font size, font style, and font effects (such as small caps or superscript). Character styles can be applied on top of paragraph styles, which means you can use a character style to override the format of a selected portion of a paragraph.
List	A list style contains paragraph number or bullet formatting for up to nine levels of an outline. All nine levels of formatting are contained in one style.

Paragraph	A paragraph style can contain any kind of paragraph formatting (such as paragraph alignment and indents) and any character formatting that you apply to an entire paragraph (such as font or font size).
Table	Table styles can contain certain elements of table, paragraph, and font formatting. You can apply formatting such as font, borders, shading, and paragraph spacing to the heading row as well as the body rows of a table, and the formatting will automatically adjust to accommodate changes in your table structure.

Paragraph

Description

Represents a paragraph in the Word document. Technically, a paragraph is a block of text that ends with a carriage return.

Signature: C#

```
public class Paragraph : Element
```

Signature: VB.NET

```
Public Class Paragraph
  Inherits Element
```

Remarks

To create a new paragraph, use Element.InsertParagraphBefore() or Element.InsertParagraphAfter(). Each of these methods take a single argument of type NamedStyle. This argument can be null. If it is null, the Normal style is used. This method can be called from the Document object because `Document` inherits from the Element class.

Examples

C#

```
//--- Insert a paragraph at the end of a new document and add some sample text
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          Paragraph paragraph = doc.InsertParagraphAfter(null);
          paragraph.InsertTextAfter("This is a new paragraph.");

          //--- Get the first paragraph of an existing document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Paragraph firstParagraph = doc.Elements(Element.Type.Paragraph)[0];
```

vb.net

```
'--- Insert a paragraph at the end of a new document and add some sample text
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim parargraph As Paragraph = doc.InsertParagraphAfter(Nothing)
          paragraph.InsertTextAfter("This is a new paragraph.")

          '--- Get the first paragraph of an existing document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim blocks() As BlockElements = doc.BlockElements
          Dim firstParagraph As Paragraph = doc.Elements(Element.Type.Paragraph)(0)
```

Properties

Name	Description

Formatting	Sets or returns a ParagraphFormatting object that represents the formatting for this paragraph (positioning, line spacing, etc).
Style	Returns or sets an NamedStyle object that represents the Style of this paragraph.

Paragraph.Formatting

Description

Sets or returns a ParagraphFormatting object that represents the formatting for this paragraph (positioning, line spacing, etc).

Signature: C#

```
public ParagraphFormatting Formatting{ get; set; }
```

Signature: VB.NET

```
Public Property Formatting() As ParagraphFormatting
```

Examples

C#

```
//--- Return ParagraphFormatting
          ParagraphFormatting formatting = oParagraph.ParagraphFormatting;

          //--- Set ParagraphFormatting
          oParagraph.ParagraphFormatting = oParagraphFormatting;
```

vb.net

```
'--- Return ParagraphFormatting
          Dim formatting As ParagraphFormatting = oParagraph.ParagraphFormatting

          '--- Set ParagraphFormatting
          oParagraph.ParagraphFormatting = oParagraphFormatting
```

Paragraph.Style

Description

Returns or sets an NamedStyle object that represents the Style of this paragraph.

Signature: C#

```
public NamedStyle Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As NamedStyle
```

Examples

C#

```
//--- Return NamedStyle
          NamedStyle oNamedStyle = oParagraph.Style;

          //--- Set NamedStyle
          oParagraph.Style = oDocument.Styles[NamedStyle.BuiltIn.Heading1];
```

vb.net

```
'--- Return NamedStyle
          Dim oNamedStyle As NamedStyle = oParagraph.Style

          '--- Set NamedStyle
          oParagraph.Style = oDocument.Styles(NamedStyle.BuilIn.Heading1)
```

ParagraphFormatting

Description

The ParagraphFormatting class is used to specify formatting that should be applied to a paragraph. It also represents the paragraph formatting of named styles in the document.

Signature: C#

```
public sealed class ParagraphFormatting
```

Signature: VB.NET

```
Public NotInheritable Class ParagraphFormatting
```

Remarks

There are two ways to obtain an instance of this class: Create a new normal paragraph formatting object using the Document.CreateParagraphFormatting method or access an existing style's paragraph formatting information using the NamedStyle.ParagraphFormatting property. The `ParagraphFormatting` object that is returned can then be used in conjunction with methods in the Element class to create paragraphs with specific formatting.

The following example demonstrates both ways of getting paragraph formatting, first by retrieving the BodyText style's paragraph formatting from the document, second by retrieving a copy of the BodyText2 style's paragraph formatting. Then, some properties are set and the formatting is applied to a new paragraph.

Examples

C#

```
//--- Get BodyText paragraph formatting from Document
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          ParagraphFormatting bodyTextFormatting =
               doc.CreateParagraphFormatting();

          //--- Set some properties
          bodyTextFormatting.PageBreakBefore = true;
          bodyTextFormatting.TextJustification =
               ParagraphFormatting.Justification.Right;

          //--- Apply it to a new paragraph
          doc.InsertParagraphAfter(null, bodyTextFormatting);

          //--- Get paragraph formatting from BodyText2 Style
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          ParagraphFormatting bodyText2Formatting =
               doc.Styles[NamedStyle.BuiltIn.BodyText2].ParagaphFormatting;

          //--- Set some properties
          bodyText2Formatting.PageBreakBefore = true;
          bodyText2Formatting.TextJustification =
               ParagraphFormatting.Justification.Right;

          //--- Apply it to a new paragraph
          doc.InsertParagraphAfter(null, bodyText2Formatting);
```

vb.net

```
'--- Get BodyText paragraph formatting from Document
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim bodyTextFormatting As ParagraphFormatting = _
               doc.CreateParagraphFormatting()

          '--- Set some properties
          bodyTextFormatting.PageBreakBefore = True
          bodyTextFormatting.TextJustification = _
               ParagraphFormatting.Justification.Right

          '--- Apply it to a new paragraph
          doc.InsertParagraphAfter(Nothing, bodyTextFormatting)

          '--- Get paragraph formatting from BodyText2 Style
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim bodyText2Formatting As ParagraphFormatting = _
               doc.Styles(NamedStyle.BuiltIn.BodyText2).ParagraphFormatting

          '--- Set some properties
          bodyText2Formatting.PageBreakBefore = True
          bodyText2Formatting.TextJustification = _
               ParagraphFormatting.Justification.Right

          '--- Apply it to a new paragraph
          doc.InsertParagraphAfter(Nothing, bodyText2Formatting)
```

Properties

Name	Description
AbsolutePositioning	Returns an AbsolutePositioning object on which you can control the absolute positioning for a paragraph.

AfterAutoSpacing	Sets or returns a `boolean` that represents if Word will automatically handle spacing below a paragraph.
AllowAutoHyphenation	Sets or returns a `boolean` representing if auto hyphening is allowed for a paragraph.
BeforeAutoSpacing	Sets or returns a `boolean` that represents if Word will automatically handle spacing above a paragraph.
KeepLinesTogether	Sets or returns a `boolean` that represents if Word attempts to keep all lines of a paragraph on the same page. (It prevents a page break within a paragraph.)
KeepWithNext	Sets or returns a `boolean` that represents if Word attempts to keep a paragraph on the same page as the next paragraph. (It prevents a page break a paragraph and the next paragraph.)
LineNumberingAllowed	Sets or returns a `boolean` that represents if Word will display line numbers next to a paragraph. This property has no effect in documents or sections with no line numbers.
LineSpacing	Returns an `int` representing the line spacing for a paragraph.
LineSpacingRule	Returns a ParagraphFormatting.LineSpacingRule object that represents the type of line spacing for a paragraph. There are 3 types: AtLeast, Exactly, and Multiple. AtLeast and Exactly use twips to specify a minimum height and exact height respectively. One twip = (1/20 pt) or (1/1440 in). Multiple specifies a spacing as a number of lines. Multiple is the most common type of spacing.
PageBreakBefore	Sets or returns a `boolean` that represents if Word will have a paragraph be the start of a new page. (It inserts a manual page break before the paragraph.)
Shading	Returns a Shading object which on which you can manipulate the shading (fill color and/or pattern) properties of a paragraph.
SpaceAfter	Sets or returns an `int` representing the amount of whitespace that should be placed below a paragraph in twips. One twip = (1/20 pt) or (1/1440 in)
SpaceBefore	Sets or returns an `int` representing the amount of whitespace that should be placed above a paragraph in twips. One twip = (1/20 pt) or (1/1440 in)
TextJustification	Sets or returns a ParagraphFormatting.Justification object that represents the text justification for a paragraph.
WidowControl	Sets or returns a `boolean` representing if Word will use Widow Control for a paragraph. When set to true, Word will not print the last line of a paragraph by itself at the top of a page (widow) or the first line of a paragraph by itself at the bottom of a page (orphan).

Methods

Name	Description
GetBorder(Border.Location)	Returns a Border object on which you can manipulate the border properties of a paragraph for a specified location. Valid Border locations are: Top, Left, Bottom, Right, Between, and Bar. Bar and Between are less obvious locations and they are explained below.
GetIndent(ParagraphFormatting.IndentLocation)	Returns an `int` representing the indent of the paragraph in twips.
SetIndent(Int32, ParagraphFormatting.IndentLocation)	Sets an `int` representing the indent of the paragraph in twips.
SetLineSpacing(Int32, ParagraphFormatting.SpacingRule)	Sets an `int` representing the line spacing for a paragraph.

Nested Classes

Name	Description
FontAlignment	Font alignments for a paragraph.
IndentLocation	Indent locations for a paragraph.
Justification	Justification types for a paragraph.
SpacingRule	Line spacing types for a paragraph. If the line spacing is not at least the height of the font used on the line then the top of the character string is truncated.

ParagraphFormatting.AbsolutePositioning

Description

Returns an AbsolutePositioning object on which you can control the absolute positioning for a paragraph.

Signature: C#

```
public AbsolutePositioning AbsolutePositioning{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property AbsolutePositioning() As AbsolutePositioning
```

Examples

C#

```
oParagraphFormatting.AbsolutePositioning.X = 100;
```

vb.net

```
oParagraphFormatting.AbsolutePositioning.X = 100
```

ParagraphFormatting.AfterAutoSpacing

Description

Sets or returns a `boolean` that represents if Word will automatically handle spacing below a paragraph.

Signature: C#

```
public boolean AfterAutoSpacing{ get; set; }
```

Signature: VB.NET

```
Public Property AfterAutoSpacing() As Boolean
```

Remarks

MS Word equivalent: Format menu > Paragraph... > Indents and Spacing tab > Spacing section > After:

Examples

C#

```
//--- Return AfterAutoSpacing
          bool afterAutoSpacing = oParagraphFormatting.AfterAutoSpacing;

          //--- Set AfterAutoSpacing
          oParagraphFormatting.AfterAutoSpacing = true;
```

vb.net

```
'--- Return AfterAutoSpacing
          Dim afterAutoSpacing As Boolean = oParagraphFormatting.AfterAutoSpacing

          '--- Set AfterAutoSpacing
          oParagraphFormatting.AfterAutoSpacing = True
```

ParagraphFormatting.AllowAutoHyphenation

Description

Sets or returns a `boolean` representing if auto hyphening is allowed for a paragraph.

Signature: C#

```
public boolean AllowAutoHyphenation{ get; set; }
```

Signature: VB.NET

```
Public Property AllowAutoHyphenation() As Boolean
```

Remarks

The default value for a new paragraph is true.

MS Word equivalent: Format menu > Paragraph... > Line and Page Breaks tab > Don't hyphenate

Examples

C#

```
//--- Return AllowAutoHyphenation
          bool allowAutoHyphenation = oParagraphFormatting.AllowAutoHyphenation;

          //--- Set AllowAutoHyphenation
          oParagraphFormatting.AllowAutoHyphenation = true;
```

vb.net

```
'--- Return AllowAutoHyphenation
          Dim allowAutoHyphenation As Boolean = oParagraphFormatting.AllowAutoHyphenation

          '--- Set AllowAutoHyphenation
          oParagraphFormatting.AllowAutoHyphenation = True
```

ParagraphFormatting.BeforeAutoSpacing

Description

Sets or returns a `boolean` that represents if Word will automatically handle spacing above a paragraph.

Signature: C#

```
public boolean BeforeAutoSpacing{ get; set; }
```

Signature: VB.NET

```
Public Property BeforeAutoSpacing() As Boolean
```

Remarks

MS Word equivalent: Format menu > Paragraph... > Indents and Spacing tab > Spacing section > Before:

Examples

C#

```
//--- Return BeforeAutoSpacing
          bool beforeAutoSpacing = oParagraphFormatting.BeforeAutoSpacing;

          //--- Set BeforeAutoSpacing
          oParagraphFormatting.BeforeAutoSpacing = true;
```

vb.net

```
'--- Return BeforeAutoSpacing
          Dim beforeAutoSpacing As Boolean = oParagraphFormatting.BeforeAutoSpacing

          '--- Set BeforeAutoSpacing
          oParagraphFormatting.BeforeAutoSpacing = True
```

ParagraphFormatting.FontAlignment

Description

Font alignments for a paragraph.

Signature: C#

```
public sealed class FontAlignment : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class FontAlignment
  Inherits TypesafeEnum
```

Values

Name	Description
Auto	Deprecated.
Centered	Deprecated.
Hanging	Deprecated.
Roman	Deprecated.
Variable	Deprecated.

ParagraphFormatting.GetBorder(Border.Location)

Description

Returns a Border object on which you can manipulate the border properties of a paragraph for a specified location. Valid Border locations are: Top, Left, Bottom, Right, Between, and Bar. Bar and Between are less obvious locations and they are explained below.

Signature: C#

```
public Border GetBorder(Location location)
```

Signature: VB.NET

```
Public Function GetBorder(ByVal location As Location) As Border
```

Parameters

location

Returns

A `Border` object on which you can manipulate the border properties of a paragraph at the given location.

Remarks

Between refers to a border to place between conforming paragraphs. Two paragraphs conform when both have borders, their brcLeft and brcRight matches, their widths are the same, they both belong to tables or both do not, and have the same AbsolutePositioning settings.

Bar refers to a border to be displayed on outside of text when facing pages are displayed.

MS Word equivalent: Format menu > Borders and Shading... > Borders tab

Examples

C#

```
oParagraphFormatting.GetBorder(Border.Location.Top).LineStyle =
                Border.LineStyle.Double;
```

vb.net

```
oParagraphFormatting.GetBorder(Border.Location.Top).LineStyle = _
                Border.LineStyle.Double
```

ParagraphFormatting.GetIndent(ParagraphFormatting.IndentLocation)

Description

Returns an `int` representing the indent of the paragraph in twips.

Signature: C#

```
public int GetIndent(IndentLocation location)
```

Signature: VB.NET

```
Public Function GetIndent(ByVal location As IndentLocation) As Integer
```

Parameters

indentLocation

A ParagraphFormatting.IndentLocation representing the indent location for which to get the indent.

Returns

An `int` that represents the indent of the paragraph in twips.

Examples

C#

```
int indent = oParagraphFormatting.GetIndent(
                ParagraphFormatting.IndentLocation.FirstLine);
```

vb.net

```
Dim indent As Integer = oParagraphFormatting.GetIndent( _
               ParagraphFormatting.IndentLocation.FirstLine)
```

ParagraphFormatting.IndentLocation

Description

Indent locations for a paragraph.

Signature: C#

```
public sealed class IndentLocation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class IndentLocation
  Inherits TypesafeEnum
```

Values

Name	Description
FirstLine	Specifies an indentation between the left margin of a page and the first line of a paragraph.
Left	Specifies an indentation between the left margin of a page and the left edge of a paragraph.
Right	Specifies an indentation between the right margin of a page and the right edge of a paragraph.

ParagraphFormatting.Justification

Description

Justification types for a paragraph.

Signature: C#

```
public sealed class Justification : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Justification
  Inherits TypesafeEnum
```

Values

Name	Description
Center	Paragraph is centered.
Left	Paragraph is left-aligned.
LeftRight	Paragraph is justified.
Right	Paragraph is right-aligned.

ParagraphFormatting.KeepLinesTogether

Description

Sets or returns a `boolean` that represents if Word attempts to keep all lines of a paragraph on the same page. (It prevents a page break within a paragraph.)

Signature: C#

```
public boolean KeepLinesTogether{ get; set; }
```

Signature: VB.NET

```
Public Property KeepLinesTogether() As Boolean
```

Remarks

MS Word equivalent: Format menu > ParagraphFormatting... > Line and Page Breaks tab > Pagination section > Keep lines together

Examples

C#

```
//--- Return KeepLinesTogether
          bool keepLinesTogether = oParagraphFormatting.KeepLinesTogether;

          //--- Set KeepLinesTogether
          oParagraphFormatting.KeepLinesTogether = true;
```

vb.net

```
'--- Return KeepLinesTogether
          Dim keepLinesTogether As Boolean = oParagraphFormatting.KeepLinesTogether

          '--- Set KeepLinesTogether
          oParagraphFormatting.KeepLinesTogether = True
```

ParagraphFormatting.KeepWithNext

Description

Sets or returns a `boolean` that represents if Word attempts to keep a paragraph on the same page as the next paragraph. (It prevents a page break a paragraph and the next paragraph.)

Signature: C#

```
public boolean KeepWithNext{ get; set; }
```

Signature: VB.NET

```
Public Property KeepWithNext() As Boolean
```

Remarks

MS Word equivalent: Format menu > Paragraph... > Line and Page Breaks tab > Pagination section > Keep with next

Examples

C#

```
//--- Return KeepWithNext
          bool keepWithNext = oParagraphFormatting.KeepWithNext;

          //--- Set KeepWithNext
          oParagraphFormatting.KeepWithNext = true;
```

vb.net

```
'--- Return KeepWithNext
          Dim keepWithNext As Boolean = oParagraphFormatting.KeepWithNext

          '--- Set KeepWithNext
          oParagraphFormatting.KeepWithNext = True
```

ParagraphFormatting.LineNumberingAllowed

Description

Sets or returns a `boolean` that represents if Word will display line numbers next to a paragraph. This property has no effect in documents or sections with no line numbers.

Signature: C#

```
public boolean LineNumberingAllowed{ get; set; }
```

Signature: VB.NET

```
Public Property LineNumberingAllowed() As Boolean
```

Remarks

MS Word equivalent: Format menu > Paragraph... > Line and Page Breaks tab > Pagination section > Suppress line numbers

Examples

C#

```
//--- Return LineNumberingAllowed
          bool lineNumberingAllowed = oParagraphFormatting.LineNumberingAllowed;

          //--- Set LineNumberingAllowed
          oParagraphFormatting.LineNumberingAllowed = true;
```

vb.net

```
'--- Return LineNumberingAllowed
          Dim lineNumberingAllowed As Boolean = oParagraphFormatting.LineNumberingAllowed

          '--- Set LineNumberingAllowed
          oParagraphFormatting.LineNumberingAllowed = True
```

ParagraphFormatting.LineSpacing

Description

Returns an `int` representing the line spacing for a paragraph.

Signature: C#

```
public int LineSpacing{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property LineSpacing() As Integer
```

Remarks

There are three types of line spacing: AtLeast, Exactly, and Multiple. AtLeast and Exactly are set in twips. One twip = (1/20 pt) or (1/1440 in). Multiple is set in number of lines. AtLeast must be between 0 pt (0 twips) and 1584 pt (31680 twips). Exactly must be between 0.7 pt (14 twips) and 1584 pt (31680 twips). Multiple must be between 1 line and 132 lines. Values beyond these for all 3 types will be adjusted automatically.

MS Word equivalent: Format menu > Paragraph... > Indents and Spacing > Spacing section > At:

The default value for this property is 2.

Examples

C#

```
//--- Return LineSpacing
          int lineSpacing = oParagraphFormatting.LineSpacing;
```

vb.net

```
'--- Return LineSpacing
          Dim lineSpacing As Integer = oParagraphFormatting.LineSpacing
```

ParagraphFormatting.LineSpacingRule

Description

Returns a ParagraphFormatting.LineSpacingRule object that represents the type of line spacing for a paragraph. There are 3 types: AtLeast, Exactly, and Multiple. AtLeast and Exactly use twips to specify a minimum height and exact height respectively. One twip = (1/20 pt) or (1/1440 in). Multiple specifies a spacing as a number of lines. Multiple is the most common type of spacing.

Signature: C#

```
public SpacingRule LineSpacingRule{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property LineSpacingRule() As SpacingRule
```

Remarks

MS Word equivalent: Format menu > ParagraphFormatting... > Indents and Spacing > Spacing section > Line spacing:

ParagraphFormatting.PageBreakBefore

Description

Sets or returns a `boolean` that represents if Word will have a paragraph be the start of a new page. (It inserts a manual page break before the paragraph.)

Signature: C#

```
public boolean PageBreakBefore{ get; set; }
```

Signature: VB.NET

```
Public Property PageBreakBefore() As Boolean
```

Remarks

MS Word equivalent: Format menu > Paragraph... > Line and Page Breaks tab > Pagination section > Page break before

Examples

C#

```
//--- Return PageBreakBefore
          bool pageBreakBefore = oParagraphFormatting.PageBreakBefore;

          //--- Set PageBreakBefore
          oParagraphFormatting.PageBreakBefore = true;
```

vb.net

```
'--- Return PageBreakBefore
          Dim pageBreakBefore As Boolean = oParagraphFormatting.PageBreakBefore

          '--- Set PageBreakBefore
          oParagraphFormatting.PageBreakBefore = True
```

ParagraphFormatting.SetIndent(Int32, ParagraphFormatting.IndentLocation)

Description

Sets an `int` representing the indent of the paragraph in twips.

Signature: C#

```
public void SetIndent(int twips, IndentLocation location)
```

Signature: VB.NET

```
Public Sub SetIndent(ByVal twips As Integer, ByVal location As IndentLocation)
```

Parameters

indentSize

An `int` representing the size in twips by which to indent.

indentLocation

A ParagraphFormatting.IndentLocation representing the indent location for which to set the indent.

Examples

C#

```
oParagraphFormatting.SetIndent(8, ParagraphFormatting.IndentLocation.FirstLine);
```

vb.net

```
oParagraphFormatting.SetIndent(8, ParagraphFormatting.IndentLocation.FirstLine)
```

ParagraphFormatting.SetLineSpacing(Int32, ParagraphFormatting.SpacingRule)

Description

Sets an `int` representing the line spacing for a paragraph.

Signature: C#

```
public void SetLineSpacing(int spacing, SpacingRule type)
```

Signature: VB.NET

```
Public Sub SetLineSpacing(ByVal spacing As Integer, ByVal type As SpacingRule)
```

Parameters

spacing

The value for the spacing. The units depend on the type parameter.

type

The type of spacing to use. See SpacingRule for a discussion on the different types of spacing.

Remarks

There are three types of line spacing: AtLeast, Exactly, and Multiple. AtLeast and Exactly are set in twips. One twip = (1/20 pt) or (1/1440 in). Multiple is set in number of lines. AtLeast must be between 0 pt (0 twips) and 1584 pt (31680 twips). Exactly must be between 0.7 pt (14 twips) and 1584 pt (31680 twips). Multiple must be between 1 line and 132 lines. Values beyond these for all 3 types will be adjusted automatically.

MS Word equivalent: Format menu > Paragraph... > Indents and Spacing > Spacing section > At:

Examples

C#

```
//--- Set LineSpacing to 2 lines
          oParagraphFormatting.SetLineSpacing(2,
               ParagraphFormatting.SpacingRule.Multiple);
```

vb.net

```
'--- Set LineSpacing to 2 lines
          oParagraphFormatting.SetLineSpacing(2, _
               ParagraphFormatting.SpacingRule.Multiple)
```

ParagraphFormatting.Shading

Description

Returns a Shading object which on which you can manipulate the shading (fill color and/or pattern) properties of a paragraph.

Signature: C#

```
public Shading Shading{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Shading() As Shading
```

Remarks

Note: The shading on a paragraph will be overridden by shading set on a character run.

MS Word equivalent: Format menu > Borders and Shading... > Shading tab

Examples

C#
`Shading shading = oParagraphFormatting.Shading;`

vb.net
`Dim shading As Shading = oParagraphFormatting.Shading`

ParagraphFormatting.SpaceAfter

Description

Sets or returns an `int` representing the amount of whitespace that should be placed below a paragraph in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int SpaceAfter{ get; set; }
```

Signature: VB.NET

```
Public Property SpaceAfter() As Integer
```

Remarks

This must be between 0 pt (0 twips) and 1584 pt (31680 twips). Values beyond these will be automatically adjusted.

MS Word equivalent: Format menu > Paragraph... > Indents and Spacing tab > Spacing section > After:

ParagraphFormatting.SpaceBefore

Description

Sets or returns an `int` representing the amount of whitespace that should be placed above a paragraph in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int SpaceBefore{ get; set; }
```

Signature: VB.NET

```
Public Property SpaceBefore() As Integer
```

Remarks

This must be between 0 pt (0 twips) and 1584 pt (31680 twips). Values beyond these will be automatically adjusted.

MS Word equivalent: Format menu > Paragraph... > Indents and Spacing > Spacing section > Before:

Examples

C#

```
//--- Return SpaceBefore
          int spaceBefore = oParagraphFormatting.SpaceBefore;

          //--- Set SpaceBefore to 12 pt
          oParagraphFormatting.SpaceBefore = 240;
```

vb.net

```
'--- Return SpaceBefore
          Dim spaceBefore As Integer = oParagraphFormatting.SpaceBefore

          '--- Set SpaceBefore to 12 pt
          oParagraphFormatting.SpaceBefore = 240
```

ParagraphFormatting.SpacingRule

Description

Line spacing types for a paragraph. If the line spacing is not at least the height of the font used on the line then the top of the character string is truncated.

Signature: C#

```
public sealed class SpacingRule : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class SpacingRule
  Inherits TypesafeEnum
```

Remarks

The default value for the ParagraphFormatting.LineSpacing property is `Multiple`

Values

Name	Description
AtLeast	Minimum height of the width between lines in twips. One twip = (1/20 pt).
Exactly	Exact height of the width between lines in twips. One twip = (1/20 pt).
Multiple	Exact height of the width between lines in number of lines. For example, enter 2 for double spacing.

ParagraphFormatting.TextJustification

Description

Sets or returns a ParagraphFormatting.Justification object that represents the text justification for a paragraph.

Signature: C#

```
public Justification TextJustification{ get; set; }
```

Signature: VB.NET

```
Public Property TextJustification() As Justification
```

Remarks

MS Word equivalent: Format menu > Paragraph... > Indents and Spacing tab > General section > Alignment:

Examples

C#

```
//--- Return Justification
          ParagraphFormatting.Justification oJustification =
               oParagraphFormatting.TextJustification;

          //--- Set Justification to Right
          oParagraphFormatting.TextJustification =
               ParagraphFormatting.Justification.Right;
```

vb.net

```
'--- Return Justification
          Dim oJustification As ParagraphFormatting.Justification = _
               oParagraphFormatting.TextJustificaiton

          '--- Set Justification
          oParagraphFormatting.TextJustification = _
               ParagraphFormatting.Justification.Right
```

ParagraphFormatting.WidowControl

Description

Sets or returns a `boolean` representing if Word will use Widow Control for a paragraph. When set to true, Word will not print the last line of a paragraph by itself at the top of a page (widow) or the first line of a paragraph by itself at the bottom of a page (orphan).

Signature: C#

```
public boolean WidowControl{ get; set; }
```

Signature: VB.NET

```
Public Property WidowControl() As Boolean
```

Remarks

The default value for a new paragraph is `true`.

MS Word equivalent: Format menu > Paragraph... > Line and Page Breaks tab > Pagination section > Widow/Orphan control

Examples

C#

```
//--- Return WidowControl
          bool widowControl = oParagraphFormatting.WidowControl;

          //--- Set WidowControl
          oParagraphFormatting.WidowControl = false;
```

vb.net

```
'--- Return WidowControl
          Dim widowControl As Boolean = oParagraphFormatting.WidowControl

          '--- Set WidowControl
          oParagraphFormatting.WidowControl = False
```

Position

Description

This class is used to represent a cursor position in the document. It sits between two characters and its position is represented as the index of the character it sits behind. A Position object can be obtained from an Element.

Signature: C#

```
public sealed class Position
```

Signature: VB.NET

```
Public NotInheritable Class Position
```

Remarks

You cannot create a new `Position` object using the `new` keyword. You must get a `Position` object using the GetPosition() method of the `Element` class (or the Document class, which inherits from `Element`). This method takes a parameter that represents the relative offset from the start of this Element to point at which to get the Position.

The following sample gets a `Position` object from the start of a `Document`.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Position pos = doc.GetPosition(0);
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim pos As Position = doc.GetPosition(0)
```

Properties

Name	Description
PositionValue	Returns an `int` that represents the absolute position in the document.

Methods

Name	Description
InsertHyperlink(String, String)	Creates and returns a Hyperlink at this Position. This hyperlink will have the specified url and displayed text.
InsertImage(String)	Inserts and returns an InlineImage at the point represented by this Position. The image that is inserted is specified by a fileName string.

InsertImage(System.IO.Stream)	Inserts and returns an InlineImage at the point represented by this Position. The image that is inserted is specified by a `Stream` object.
InsertList(Boolean)	Creates and returns an empty List at the point represented by this Position. By passing a boolean, it can be a numbered list (true) or a bulleted list (false).
InsertMergeField(String, String)	Creates and returns a MergeField at this Position. This merge field will have the specified name and contents.
InsertParagraph(NamedStyle)	Creates and returns an empty Paragraph at the point represented by this Position, formatted with the specified style. If this Position is in the middle of a paragraph the containing paragraph will be split.
InsertSection()	Creates and returns an empty Section at the point represented by this Position. If this Position is in the middle of a section then the containing section will be split.
InsertTable(Int32, Int32)	Creates and returns a Table at the point represented by this Position. The table will contain the specified number of rows and columns.
InsertText(String, Boolean)	Creates and returns a CharacterRun at the point represented by this Position. The text to be inserted is specified as well as whether this should be a new character run or part of the previous one.

Position.InsertHyperlink(String, String)

Description

Creates and returns a Hyperlink at this Position. This hyperlink will have the specified url and displayed text.

Signature: C#

```
public Hyperlink InsertHyperlink(System.String url, System.String display)
```

Signature: VB.NET

```
Public Function InsertHyperlink(ByVal url As String, ByVal display As String) As Hyperlink
```

Parameters

url

A `String` representing the url to link to.

display

A `String` representing the text that will be displayed in the document for this hyperlink.

Returns

A `Hyperlink` object representing the newly created hyperlink.

Exceptions

System.Exception

Examples

C#

```
Hyperlink link = e.InsertHyperlink("http://www.softartisans.com", "SoftArtisans");
```

vb.net

```
Dim link As Hyperlink = e.InsertHyperlink("http://www.softartisans.com", "SoftArtisans")
```

Position.InsertImage

Overload	Description
InsertImage(String)	Inserts and returns an InlineImage at the point represented by this Position. The image that is inserted is specified by a fileName string.
InsertImage(System.IO.Stream)	Inserts and returns an InlineImage at the point represented by this Position. The image that is inserted is specified by a `Stream` object.

Position.InsertImage(String)

Description

Inserts and returns an InlineImage at the point represented by this Position. The image that is inserted is specified by a fileName string.

Signature: C#

```
public InlineImage InsertImage(System.String fileName)
```

Signature: VB.NET

```
Public Function InsertImage(ByVal fileName As String) As InlineImage
```

Parameters

fileName

A `String` representing the file anme of the image to insert.

Returns

An `InlineImage` object representing the inserted image.

Exceptions

System.Exception

Examples

C#

```
InlineImage image = pos.InsertImage("C:\\sample.jpg");
```

vb.net

```
Dim image As InlineImage = pos.InsertImage("C:\sample.jpg")
```

Position.InsertImage(System.IO.Stream)

Description

Inserts and returns an InlineImage at the point represented by this Position. The image that is inserted is specified by a `Stream` object.

Signature: C#

```
public InlineImage InsertImage(System.IO.Stream stream)
```

Signature: VB.NET

```
Public Function InsertImage(ByVal stream As System.IO.Stream) As InlineImage
```

Parameters

stream

A `Stream` object that the image can be read from.

Returns

An `InlineImage` object representing the inserted image.

Exceptions

System.Exception

If the stream does not contain a supported image format see InlineImage.ImageFormat for a list of supported formats.

Position.InsertList(Boolean)

Description

Creates and returns an empty List at the point represented by this Position. By passing a boolean, it can be a numbered list (true) or a bulleted list (false).

Signature: C#

```
public List InsertList(boolean numbered)
```

Signature: VB.NET

```
Public Function InsertList(ByVal numbered As Boolean) As List
```

Parameters

numbered

Returns

A `List` object representing the newly created empty list.

Examples

C#

```
List lst = pos.InsertList(true);
```

vb.net

```
Dim lst As List = pos.InsertList(True)
```

Position.InsertMergeField(String, String)

Description

Creates and returns a MergeField at this Position. This merge field will have the specified name and contents.

Signature: C#

```
public MergeField InsertMergeField(System.String fieldName, System.String display)
```

Signature: VB.NET

```
Public Function InsertMergeField(ByVal fieldName As String, ByVal display As String) As MergeField
```

Parameters

fieldName

A `String` representing the name of this merge field.

display

A `String` representing the text that will displayed for this merge field in the document.

Returns

A `MergeField` object representing the newly created merge field.

Exceptions

System.Exception

Examples

C#

```
MergeField mfield = e.InsertMergeField("MyField", "ThisNameIsDisplayed");
```

vb.net

```
Dim mfield As MergeField = e.InsertMergeField("MyField", "ThisNameIsDisplayed")
```

Position.InsertParagraph(NamedStyle)

Description

Creates and returns an empty Paragraph at the point represented by this Position, formatted with the specified style. If this Position is in the middle of a paragraph the containing paragraph will be split.

Signature: C#

```
public Paragraph InsertParagraph(NamedStyle style)
```

Signature: VB.NET

```
Public Function InsertParagraph(ByVal style As NamedStyle) As Paragraph
```

Parameters

style

The base style of this paragraph. The base style also affects the character style of any text added to this paragraph. If this is null, the style will be "Normal".

Returns

A newly created `Paragraph`. Even though the paragraph is empty there will always be at least a single carriage return.

Examples

C#

```
Paragraph p = pos.InsertParagraph(oDocument.Styles[NamedStyle.BuiltIn.Normal]);
```

vb.net

```
Dim p As Paragraph = pos.InsertParagraph(oDocument.Styles(NamedStyle.BuiltIn.Normal))
```

Position.InsertSection()

Description

Creates and returns an empty Section at the point represented by this Position. If this Position is in the middle of a section then the containing section will be split.

Signature: C#

```
public Section InsertSection()
```

Signature: VB.NET

```
Public Function InsertSection() As Section
```

Returns

A newly created `Section`.

Examples

C#

```
Section sect = pos.InsertSection();
```

vb.net

```
Dim sect As Section = pos.InsertSection()
```

Position.InsertTable(Int32, Int32)

Description

Creates and returns a Table at the point represented by this Position. The table will contain the specified number of rows and columns.

Signature: C#

```
public Table InsertTable(int rows, int columns)
```

Signature: VB.NET

```
Public Function InsertTable(ByVal rows As Integer, ByVal columns As Integer) As Table
```

Parameters

rows

An `int` representing the number of rows the table will have.

columns

An `int` representing the number of columns the table will have.

Returns

A `Table` object representing the newly created table.

Examples

C#

```
Table tbl = pos.InsertTable(2, 3);
```

vb.net

```
Dim tbl As Table = pos.InsertTable(2, 3)
```

Position.InsertText(String, Boolean)

Description

Creates and returns a CharacterRun at the point represented by this Position. The text to be inserted is specified as well as whether this should be a new character run or part of the previous one.

Signature: C#

```
public CharacterRun InsertText(System.String text, boolean newRun)
```

Signature: VB.NET

```
Public Function InsertText(ByVal text As String, ByVal newRun As Boolean) As CharacterRun
```

Parameters

text

A `String` representing the text to insert.

newRun

A `boolean` representing if this should have default formatting and become its own character run (true) or if it should assume the formatting of the first character in this element, becoming a part of that first character run.

Returns

The `CharacterRun` containing the inserted text.

Examples

C#

```
CharacterRun charRun = pos.InsertText("Here is some text.", true);
```

vb.net

```
Dim charRun As CharacterRun = pos.InsertText("Here is some text.", True)
```

Position.PositionValue

Description

Returns an `int` that represents the absolute position in the document.

Signature: C#

```
public int PositionValue{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PositionValue() As Integer
```

Examples

C#

```
int positionValue = pos.PositionValue;
```

vb.net

```
Dim positionValue As Integer = pos.PositionValue
```

SearchMatch

Description

A SearchMatch object represents a match in the document found by using the method Element.Search.

Signature: C#

```
public sealed class SearchMatch
```

Signature: VB.NET

```
Public NotInheritable Class SearchMatch
```

Examples

C#

```
//--- Required for IEnumerator
          using System.Collections;
          ...
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"c:\myDoc.doc");

          //--- Search a document for social security numbers, or any series of
          //--- nine digits, and remove them. The question mark after each dash
          //--- makes the presence of the dash optional.
          IEnumerator searcherator = doc.Search(@"\d\d\d-?\d\d-?\d\d\d\d").GetEnumerator();
          while (searcherator.MoveNext())
          {
               SearchMatch match = (SearchMatch)searcherator.Current;
               match.Element.DeleteElement();
          }
```

vb.net

```
'--- Required for IEnumerator
          Imports System.Collections
          ...
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("c:\myDoc.doc")

          '--- Search a document for social security numbers, or any series of
          '--- nine digits, and remove them. The question mark after each dash
          '--- makes the presence of the dash optional.
          Dim searcherator As IEnumerator = doc.Search(@"\d\d\d-?\d\d-?\d\d\d\d").GetEnumerator()
          While searcherator.MoveNext()
               Dim match As SearchMatch = searcherator.Current
               match.Element.DeleteElement()
          End While
```

Properties

Name	Description
Element	Returns an Element object representing the current text match.
GroupCount	Returns the number of groups found in the current SearchMatch. Regular expression groups are created by adding parentheses to the regular expression. This method should always return at least 1.

Methods

Name	Description
Group(Int32)	Returns the string that matches the specified group. The first group is always the string that matches the entire regular expression. Groups are ordered by the left-to-right order of the left parentheses of the group.
Replace(String)	Replaces this match with the specified string.

SearchMatch.Element

Description

Returns an Element object representing the current text match.

Signature: C#

```
public Element Element{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Element() As Element
```

Examples

C#

```
//--- Search for social security numbers using regular expressions and delete them.
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\somedoc.doc");
          SearchMatch[] matches = doc.Search("\s\d\d\d-?\d\d-?\d\d\d\d\s");
          foreach(SearchMatch match in matches)
          {
             match.Element.Delete();
          }
```

vb.net

```
'--- Search for social security numbers using regular expressions and delete them.
          Dim app As New WordApplication()
          Document doc = app.Open("C:\somedoc.doc")
          Dim matches As SearchMatch() = doc.Search("\s\d\d\d-?\d\d-?\d\d\d\d\s")
          For Each match As SearchMatch In matches
               match.Element.Delete()
          Loop
```

SearchMatch.Group(Int32)

Description

Returns the string that matches the specified group. The first group is always the string that matches the entire regular expression. Groups are ordered by the left-to-right order of the left parentheses of the group.

Signature: C#

```
public System.String Group(int group)
```

Signature: VB.NET

```
Public Function Group(ByVal group As Integer) As String
```

Parameters

group

The 0-based index of the group for which to return a match.

Returns

The String that matches the specified group.

Examples

C#

```
string firstGroupMatch = searchMatch.Group(0);
```

vb.net

```
Dim firstGroupMatch As String = searchMatch.Group(0)
```

SearchMatch.GroupCount

Description

Returns the number of groups found in the current SearchMatch. Regular expression groups are created by adding parentheses to the regular expression. This method should always return at least 1.

Signature: C#

```
public int GroupCount{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property GroupCount() As Integer
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\somedoc.doc");
          IEnumerator searcherator = doc.Search("\s\d\d\d-?\d\d-?\d\d\d\d\s");
          while (searcherator.MoveNext())
          {
               SearchMatch match = (SearchMatch)searcherator.Current;
               int numMatchGroups = match.GroupCount;
               match.Element.Delete();
          }
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\somedoc.doc")
          Dim searcherator As IEnumerator = doc.Search("\s\d\d\d-?\d\d-?\d\d\d\d\s")
          While searcherator.MoveNext()
               Dim match As SearchMatch = searcherator.Current
               Dim numMatchGroups As Integer = match.GroupCount
               match.Element.Delete()
          End While
```

SearchMatch.Replace(String)

Description

Replaces this match with the specified string.

Signature: C#

```
public void Replace(System.String replaceWith)
```

Signature: VB.NET

```
Public Sub Replace(ByVal replaceWith As String)
```

Parameters

replaceWith

The replacement string.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\somedoc.doc");
          IEnumerator searcherator = doc.Search("crimson");
          while (searcherator.MoveNext())
          {
               SearchMatch match = (SearchMatch)searcherator.Current;
               match.Replace("red");
          }
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\somedoc.doc")
          Dim searcherator As IEnumerator = doc.Search("crimson")
          While searcherator.MoveNext()
               Dim match As SearchMatch = searcherator.Current
               match.Replace("red")
          End While
```

Section

Description

This class is used to represent a section of the document. Many documents will only contain one section. More complex documents will divide the document into multiple sections. A section can be roughly correlated to a chapter in a book. Each section can have different formatting settings, including:

Signature: C#

```
public sealed class Section : Element
```

Signature: VB.NET

```
Public NotInheritable Class Section
  Inherits Element
```

Remarks

To create a new section, use Element.CreateSectionBefore().

To get an existing section, use Document.Sections.

The following sample shows you how to get the add a new section to a document and how to get the first section of a document.

Examples

C#

```
//--- Add a new section to a document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Section sect = doc.CreateSectionAfter();

          //--- Get the first section of a document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Section sec1 = doc.Sections[0];
```

vb.net

```
'--- Add a new section to a document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim sect As Section = doc.CreateSectionAfter()

          '--- Get the first section of a document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim sect As Section = doc.Sections(0)
```

Properties

Name	Description
BorderAlwaysInFront	Sets or returns a `boolean` that represents whether or not page borders for this section will always display in front of any objects it may overlap.
BorderAppliedTo	Sets or returns a BorderArea object that represents where the page border will be applied to.
BorderOffsetsFromPage	Sets or returns a `boolean` that represents whether or not the padding for this section's page border will be measured from the edge of the page or from the text it surrounds.
BottomMargin	Sets or returns an `int` that represents the vertical distance of the bottom margin from the bottom edge of the page for this section. One twip = (1/20 pt) or (1/1440 in)
Break	Sets or returns a Section.BreakType object that represents the type of break (before) this section. There are 5 types of breaks: Continuous, Column, Page, EvenPage, and OddPage.
FirstLineNum	Sets or returns an `int` that represents the number that the line numbering starts at for this section.
FirstPageNum	Sets or returns an `int` that represents the number that the page numbering starts at for this section.

FooterY	Sets or returns an `int` that represents the y coordinate of the footer measured from the top edge of the page. One twip = (1/20 pt) or (1/1440 in)
GutterWidth	Sets or returns an `int` that represents the gutter width for this section. The gutter adds space to the margin. Gutters are used when a document needs additional space to allow for binding. One twip = (1/20 pt) or (1/1440 in)
HasTitlePage	Sets or returns a `boolean` that represents whether or not this section has a title page. This can affect page numbering and the appearance of headers and footers.
HeaderY	Sets or returns an `int` that represents the y coordinate of the header measured from the top edge of the page. One twip = (1/20 pt) or (1/1440 in)
LeftMargin	Sets or returns an `int` that represents the horizontal distance of the left margin from the left edge of the page for this section. One twip = (1/20 pt) or (1/1440 in)
LineNumberingType	Sets or returns a Section.LineNumbering object that represents the line numbering restart policy for this section. There are 3 options:
LineNumberX	Sets or returns an `int` that represents the x position of line numbers in twips. One twip = (1/20 pt) or (1/1440 in)
LineNumModulus	Sets or returns an `int` that represents the line numbering modulus. This setting is used to set whether line numbers display on every line, every 2nd line, etc. Setting this to 0 (zero) will display no line numbers. Setting this to 1 will display a number by every line. Setting this to 2 will display a number by every 2nd line, etc.
LinesBetween	Sets or returns a `boolean` that represents whether or not Word will draw vertical lines between columns.
NumColumns	Sets or returns an `int` that represents the number of columns in this section.
PageHeight	Sets or returns an `int` that represents the page height for this section in twips. One twip = (1/20 pt) or (1/1440 in)
PageNumberFormat	Sets or returns a Section.NumberFormat object that represents the page number format for this section. There are 5 type of page number formats:
PageOrientation	Sets or returns a Section.Orientation object that represents the page orientation for this section. Possible values: Portrait, Landscape, Mixed
PageSize	Sets or returns the size of printed pages.
PageWidth	Sets or returns an `int` that represents the page width for this section in twips. One twip = (1/20 pt) or (1/1440 in)
RestartPageNum	Sets or returns a `boolean` that represents whether or not page numbering will restart at the beginning of this section.
RightMargin	Sets or returns an `int` that represents the horizontal distance of the right margin from the right edge of the page for this section. One twip = (1/20 pt) or (1/1440 in)
TopMargin	Sets or returns an `int` that represents the vertical distance of the top margin from the top edge of the page for this section. One twip = (1/20 pt) or (1/1440 in)
Unlocked	Sets or returns a `boolean` that represents whether or not this section is unlocked.

Methods

Name	Description
GetBorder(Border.Location)	Returns a Border object that represents the border of this section at the specified location. You can use the returned object to manipulate the borders for this section. Valid locations: Top, Left, Bottom, Right
GetFooter(Section.HeaderFooterType)	Returns an Element object that represents the footer of the specified type for this section. The footer is returned as a editable element. There are 4 types of footers:
GetHeader(Section.HeaderFooterType)	Returns an Element object that represents the header of the specified type for this section. The header is returned as a editable element. There are 4 types of headers:

Nested Classes

Name	Description
BorderArea	Areas that a border can be applied to for a section.
BreakType	Types of breaks that can preceed a section.

HeaderFooterType	Types of headers and footers.
LineNumbering	Line numbering settings for a section.
NumberFormat	Formats for page numbers in a section.
Orientation	Page orientations for a section.
PagePaperSize	Page paper sizes that can be applied to a section. All native sizes are supported, and additional sizes are provided

Section.BorderAlwaysInFront

Description

Sets or returns a `boolean` that represents whether or not page borders for this section will always display in front of any objects it may overlap.

Signature: C#

```
public boolean BorderAlwaysInFront{ get; set; }
```

Signature: VB.NET

```
Public Property BorderAlwaysInFront() As Boolean
```

Section.BorderAppliedTo

Description

Sets or returns a BorderArea object that represents where the page border will be applied to.

Signature: C#

```
public BorderArea BorderAppliedTo{ get; set; }
```

Signature: VB.NET

```
Public Property BorderAppliedTo() As BorderArea
```

Section.BorderArea

Description

Areas that a border can be applied to for a section.

Signature: C#

```
public sealed class BorderArea : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BorderArea
  Inherits TypesafeEnum
```

Values

Name	Description
AllButFirst	Apply border to all but the first page in the section
AllPages	Apply border to all pages in section

Document	Apply border to all pages in section
FirstPage	Apply border to first page in section

Section.BorderOffsetsFromPage

Description

Sets or returns a `boolean` that represents whether or not the padding for this section's page border will be measured from the edge of the page or from the text it surrounds.

Signature: C#

```
public boolean BorderOffsetsFromPage{ get; set; }
```

Signature: VB.NET

```
Public Property BorderOffsetsFromPage() As Boolean
```

Section.BottomMargin

Description

Sets or returns an `int` that represents the vertical distance of the bottom margin from the bottom edge of the page for this section. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int BottomMargin{ get; set; }
```

Signature: VB.NET

```
Public Property BottomMargin() As Integer
```

Remarks

MS Word equivalent: File menu > Page Setup... > Margins tab > Bottom: -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return BottomMargin
          int bottomMargin = oSection.BottomMargin;

          //--- Set BottomMargin to 1 inch
          oSection.BottomMargin = 1440;
```

vb.net

```
'--- Return BottomMargin
          Dim bottomMargin As Integer = oSection.BottomMargin

          '--- Set BottomMargin to 1 inch
          oSection.BottomMargin = 1440
```

Section.Break

Description

Sets or returns a Section.BreakType object that represents the type of break (before) this section. There are 5 types of breaks: Continuous, Column, Page, EvenPage, and OddPage.

Signature: C#

```
public BreakType Break{ get; set; }
```

Signature: VB.NET

```
Public Property Break() As BreakType
```

Remarks

MS Word equivalent: File menu > Page Setup... > Layout tab > Section section > Section start:

Examples

C#

```
//--- Return Break
          Section.BreakType oBreak = oSection.Break;

          //--- Set BreakType
          oSection.Break = Section.BreakType.Column;
```

vb.net

```
'--- Return Break
          Dim oBreakType As Section.BreakType = oSection.Break

          '--- Set BreakType
          oSection.Break = Section.BreakType.Column
```

Section.BreakType

Description

Types of breaks that can preceed a section.

Signature: C#

```
public sealed class BreakType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BreakType
  Inherits TypesafeEnum
```

Values

Name	Description
Continuous	Start this section on the first line after the previous section
EvenPage	Start this section on the first even page after the previous section
OddPage	Start this section on the first odd page after the previous section
Page	Start this section on the page after the previous section

Section.FirstLineNum

Description

Sets or returns an `int` that represents the number that the line numbering starts at for this section.

Signature: C#

```
public int FirstLineNum{ get; set; }
```

Signature: VB.NET

```
Public Property FirstLineNum() As Integer
```

Remarks

MS Word equivalent: File menu > Page Setup... > Layout tab > Line Numbers... button > Start at: -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return FirstLineNum
          int firstLineNum = oSection.FirstLineNum;

          //--- Set FirstLineNum
          oSection.FirstLineNum = 2;
```

vb.net

```
'--- Return FirstLineNum
          Dim firstLineNum As Integer = oSection.FirstLineNum

          '--- Set FirstLineNum
          oSection.FirstLineNum = 2
```

Section.FirstPageNum

Description

Sets or returns an `int` that represents the number that the page numbering starts at for this section.

Signature: C#

```
public int FirstPageNum{ get; set; }
```

Signature: VB.NET

```
Public Property FirstPageNum() As Integer
```

Remarks

MS Word equivalent: Insert menu > Page Numbers... > Format... button > Start at:

Examples

C#

```
//--- Return FirstPageNum
          int firstPageNum = oSection.FirstPageNum;

          //--- Set FirstPageNum
          oSection.FirstPageNum = 2;
```

vb.net

```
'--- Return FirstPageNum
          Dim firstPageNum As Integer = oSection.FirstPageNum

          '--- Set FirstPageNum
          oSection.FirstPageNum = 2
```

Section.FooterY

Description

Sets or returns an `int` that represents the y coordinate of the footer measured from the top edge of the page. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int FooterY{ get; set; }
```

Signature: VB.NET

```
Public Property FooterY() As Integer
```

Examples

C#

```
//--- Return FooterY
          int footerY = oSection.FooterY;

          //--- Set FooterY to 1 inch
          oSection.FooterY = 1440;
```

vb.net

```
'--- Return FooterY
          Dim footerY As Integer = oSection.FooterY

          '--- Set FooterY to 1 inch
          oSection.FooterY = 1440
```

Section.GetBorder(Border.Location)

Description

Returns a Border object that represents the border of this section at the specified location. You can use the returned object to manipulate the borders for this section. Valid locations: Top, Left, Bottom, Right

Signature: C#

```
public Border GetBorder(Location location)
```

Signature: VB.NET

```
Public Function GetBorder(ByVal location As Location) As Border
```

Parameters

location

Returns

A `Border` object that represents the border of this section at the specified location.

Remarks

MS Word equivalent: Format menu > Borders and shading... > Borders tab

Examples

C#

```
Border oBorder = oSection.GetBorder(Border.Location.Top);
```

vb.net

```
Dim oBorder As Border = oSection.GetBorder(Border.Location.Top)
```

Section.GetFooter(Section.HeaderFooterType)

Description

Returns an Element object that represents the footer of the specified type for this section. The footer is returned as a editable element. There are 4 types of footers:

Signature: C#

```
public Element GetFooter(HeaderFooterType type)
```

Signature: VB.NET

```
Public Function GetFooter(ByVal type As HeaderFooterType) As Element
```

Parameters

type

A Section.HeaderFooterType object that represents the type of footer to return.

Returns

An `Element` object that represents the footer of the specified type for this section.

Examples

C#

```
Element oElement = oSection.GetFooter(Section.HeaderFooterType.All);
```

vb.net

```
Dim oElement As Element = oSection.GetFooter(Section.HeaderFooterType.All)
```

Section.GetHeader(Section.HeaderFooterType)

Description

Returns an Element object that represents the header of the specified type for this section. The header is returned as a editable element. There are 4 types of headers:

Signature: C#

```
public Element GetHeader(HeaderFooterType type)
```

Signature: VB.NET

```
Public Function GetHeader(ByVal type As HeaderFooterType) As Element
```

Parameters

type

A Section.HeaderFooterType object that represents the type of header to return.

Returns

An `Element` object that represents the header of the specified type for this section.

Examples

C#

```
Element oElement = oSection.GetHeader(Section.HeaderFooterType.All);
```

vb.net

```
Dim oElement As Element = oSection.GetHeader(Section.HeaderFooterType.All)
```

Section.GutterWidth

Description

Sets or returns an `int` that represents the gutter width for this section. The gutter adds space to the margin. Gutters are used when a document needs additional space to allow for binding. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int GutterWidth{ get; set; }
```

Signature: VB.NET

```
Public Property GutterWidth() As Integer
```

Remarks

Gutter width may not be smaller than 0" (0 twips) or larger than 22" (31680 twips). Values beyond these will be automatically adjusted.

MS Word equivalent: File menu > Page Setup... > Margins tab > Gutter: -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return GutterWidth
          int gutterWidth = oSection.GutterWidth;

          //--- Set GutterWidth to 1/2 inch
          oSection.GutterWidth = 720;
```

vb.net

```
'--- Return GutterWidth
          Dim gutterWidth As Integer = oSection.GutterWidth

          '--- Set GutterWidth to 1/2 inch
          oSection.GutterWidth = 720
```

Section.HasTitlePage

Description

Sets or returns a `boolean` that represents whether or not this section has a title page. This can affect page numbering and the appearance of headers and footers.

Signature: C#

```
public boolean HasTitlePage{ get; set; }
```

Signature: VB.NET

```
Public Property HasTitlePage() As Boolean
```

Examples

C#

```
//--- Return HasTitlePage
          bool hasTitlePage = oSection.HasTitlePage;


          //--- Set HasTitlePage
          oSection.HasTitlePage = true;
```

vb.net

```
'--- Return HasTitlePage
          Dim hasTitlePage As Boolean = oSection.HasTitlePage

          '--- Set HasTitlePage
          oSection.HasTitlePage = True
```

Section.HeaderFooterType

Description

Types of headers and footers.

Signature: C#

```
public sealed class HeaderFooterType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class HeaderFooterType
  Inherits TypesafeEnum
```

Values

Name	Description
All	Header or footer that is seen over all pages in a section. Has the lowest precedence of any header or footer type.
Even	Header or footer that is seen only on even pages. Takes precedence over "All" headers and footers
FirstPage	Header or footer that is seen on only the first page. Takes precedence over "Odd" headers and footers.
Odd	Header or footer that is seen only on odd pages. Takes precedence over "All" headers and footers

Section.HeaderY

Description

Sets or returns an `int` that represents the y coordinate of the header measured from the top edge of the page. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int HeaderY{ get; set; }
```

Signature: VB.NET

```
Public Property HeaderY() As Integer
```

Examples

C#

```
//--- Return HeaderY
          int headerY = oSection.HeaderY;

          //--- Set HeaderY to 1 inch
          oSection.HeaderY = 1440;
```

vb.net

```
'--- Return HeaderY
          Dim headerY As Integer = oSection.HeaderY

          '--- Set HeaderY to 1 inch
          oSection.HeaderY = 1440
```

Section.LeftMargin

Description

Sets or returns an `int` that represents the horizontal distance of the left margin from the left edge of the page for this section. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int LeftMargin{ get; set; }
```

Signature: VB.NET

```
Public Property LeftMargin() As Integer
```

Remarks

MS Word equivalent: File menu > Page Setup... > Margins tab > Left: -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return LeftMargin
          int leftMargin = oSection.LeftMargin;

          //--- Set LeftMargin to 1 inch
          oSection.LeftMargin = 1440;
```

vb.net

```
'--- Return LeftMargin
          Dim leftMargin As Integer = oSection.LeftMargin

          '--- Set LeftMargin to 1 inch
          oSection.LeftMargin = 1440
```

Section.LineNumbering

Description

Line numbering settings for a section.

Signature: C#

```
public sealed class LineNumbering : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class LineNumbering
  Inherits TypesafeEnum
```

Values

Name	Description
Continue	Continue from the last line number
PerPage	Restart for each page
Restart	Restart the numbers on this section

Section.LineNumberingType

Description

Sets or returns a Section.LineNumbering object that represents the line numbering restart policy for this section. There are 3 options:

Signature: C#

```
public LineNumbering LineNumberingType{ get; set; }
```

Signature: VB.NET

```
Public Property LineNumberingType() As LineNumbering
```

Remarks

This setting defaults to PerPage (line numbering restarts for each page).

MS Word equivalent: File > Page Setup... > Layout tab > Preview section (make sure Apply To: This Section is selected) > Line Numbers... button > Numbering section

Examples

C#

```
//--- Return LineNumbering
          Section.LineNumbering oLineNumbering = oSection.LineNumberingType;

          //--- Set LineNumbering
          oSection.LineNumberingType = Section.LineNumbering.Restart;
```

vb.net

```
'--- Return LineNumbering
          Dim oLineNumbering As Section.LineNumbering = oSection.LineNumberingType

          '--- Set LineNumbering
          oSection.LineNumberingType = Section.LineNumbering.Restart
```

Section.LineNumberX

Description

Sets or returns an `int` that represents the x position of line numbers in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int LineNumberX{ get; set; }
```

Signature: VB.NET

```
Public Property LineNumberX() As Integer
```

Remarks

MS Word equivalent: File > Page Setup... > Layout tab > Preview section (make sure Apply To: This Section is selected) > Line Numbers... button > From text:

Examples

C#

```
//--- Return LineNumberX
          int lineNumberX = oSection.LineNumberX;

          //--- Set LineNumberX to .2 inches
          oSection.LineNumberX = 288;
```

vb.net

```
'--- Return LineNumberX
          Dim lineNumberX As Integer = oSection.LineNumberX

          '--- Set LineNumberX to .2 inches
          oSection.LineNumberX = 288
```

Section.LineNumModulus

Description

Sets or returns an `int` that represents the line numbering modulus. This setting is used to set whether line numbers display on every line, every 2nd line, etc. Setting this to 0 (zero) will display no line numbers. Setting this to 1 will display a number by every line. Setting this to 2 will display a number by every 2nd line, etc.

Signature: C#

```
public int LineNumModulus{ get; set; }
```

Signature: VB.NET

```
Public Property LineNumModulus() As Integer
```

Remarks

MS Word equivalent: File > Page Setup... > Layout tab > Preview section (make sure Apply To: This Section is selected) > Line Numbers... button > Count by:

Examples

C#

```
//--- Return LineNumModulus
          int lineNumModulus = oSection.LineNumModulus;

          //--- Set LineNumModulus
          oSection.LineNumModulus = 2;
```

vb.net

```
'--- Return LineNumModulus
          Dim lineNumModulus As Integer = oSection.LineNumModulus

          '--- Set LineNumModulus
          oSection.LineNumModulus = 2
```

Section.LinesBetween

Description

Sets or returns a `boolean` that represents whether or not Word will draw vertical lines between columns.

Signature: C#

```
public boolean LinesBetween{ get; set; }
```

Signature: VB.NET

```
Public Property LinesBetween() As Boolean
```

Remarks

MS Word equivalent: Format menu > Columns... > Line between

Examples

C#

```
//--- Return LinesBetween
          bool linesBetween = oSection.LinesBetween;

          //--- Set LinesBetween
          oSection.LinesBetween = true;
```

vb.net

```
'--- Return LinesBetween
          Dim linesBetween As Boolean = oSection.LinesBetween

          '--- Set LinesBetween
          oSection.LinesBetween = True
```

Section.NumberFormat

Description

Formats for page numbers in a section.

Signature: C#

```
public sealed class NumberFormat : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class NumberFormat
  Inherits TypesafeEnum
```

Values

Name	Description
Arabic	Standard Arabic Numerals
LetterLower	Lower-Case Roman Alphabet
LetterUpper	Upper-Case Roman Alphabet
RomanLower	Lower-Case Roman Numerals
RomanUpper	Upper-Case Roman Numerals

Section.NumColumns

Description

Sets or returns an `int` that represents the number of columns in this section.

Signature: C#

```
public int NumColumns{ get; set; }
```

Signature: VB.NET

```
Public Property NumColumns() As Integer
```

Examples

C#

```
//--- Return NumColumns
          int numColumns = oSection.NumColumns;

          //--- Set NumColumns
          oSection.NumColumns = 2;
```

vb.net

```
'--- Return NumColumns
          Dim numColumns As Integer = oSection.NumColumns

          '--- Set NumColumns
          oSection.NumColumns = 2
```

Section.Orientation

Description

Page orientations for a section.

Signature: C#

```
public sealed class Orientation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Orientation
  Inherits TypesafeEnum
```

Values

Name	Description
Landscape	Landscape orientation, longer horizontal dimension
Mixed	
Portrait	Portrait orientation, longer vertical dimension

Section.PageHeight

Description

Sets or returns an `int` that represents the page height for this section in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int PageHeight{ get; set; }
```

Signature: VB.NET

```
Public Property PageHeight() As Integer
```

Remarks

Page height may not be smaller than .1" (144 twips) or larger than 22" (31680 twips). Values beyond these will be automatically adjusted.

MS Word equivalent: File menu > Page Setup... > Paper tab > Paper size: section > Height: -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return PageHeight
          int pageHeight = oSection.PageHeight;

          //--- Set PageHeight to 11 inches
          oSection.PageHeight = 15840;
```

vb.net

```
'--- Return PageHeight
          Dim pageHeight As Integer = oSection.PageHeight

          '--- Set PageHeight to 11 inches
          oSection.PageHeight = 15840
```

Section.PageNumberFormat

Description

Sets or returns a Section.NumberFormat object that represents the page number format for this section. There are 5 type of page number formats:

Signature: C#

```
public NumberFormat PageNumberFormat{ get; set; }
```

Signature: VB.NET

```
Public Property PageNumberFormat() As NumberFormat
```

Remarks

MS Word equivalent: Insert menu > Page Numbers... > Format... button > Number format:

Examples

C#

```
//--- Return PageNumberFormat
          Section.NumberFormat oPageNumberFormat = oSection.PageNumberFormat;

          //--- Set PageNumberFormat
          oSection.PageNumberFormat = Section.NumberFormat.Arabic;
```

vb.net

```
'--- Return PageNumberFormat
          Dim oPageNumberFormat As Section.NumberFormat = oSection.PageNumberFormat

          '--- Set PageNumberFormat
          oSection.PageNumberFormat = Section.NumberFormat.Arabic
```

Section.PageOrientation

Description

Sets or returns a Section.Orientation object that represents the page orientation for this section. Possible values: Portrait, Landscape, Mixed

Signature: C#

```
public Orientation PageOrientation{ get; set; }
```

Signature: VB.NET

```
Public Property PageOrientation() As Orientation
```

Remarks

MS Word equivalent: File menu > Page Setup... > Margins tab > Orientation section -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return PageOrientation
          Section.Orientation oPageOrientation = oSection.PageOrientation;

          //--- Set PageOrientation
          oSection.PageOrientation = Section.Orientation.Landscape;
```

vb.net

```
'--- Return PageOrientation
          Dim oPageOrientation As Section.Orientation = oSection.PageOrientation

          '--- Set PageOrientation
          oSection.PageOrientation = Section.Orientation.Landscape
```

Section.PagePaperSize

Description

Page paper sizes that can be applied to a section. All native sizes are supported, and additional sizes are provided

Signature: C#

```
public sealed class PagePaperSize : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class PagePaperSize
  Inherits TypesafeEnum
```

Values

Name	Description
A3	ISO A3; 11.69" x 16.54"
A4	ISO A4; 8.27" x 11.69
A4Small	Custom; 7.47" x 10.85"
A5	ISO A5; 5.83" x 8.27"
B4	ISO B4; 9.84" x 13.94"
B5	ISO B5; 7.17" x 10.12"
CSheet	ANSI C; 17" x 22"
Custom	There are known issues with this enumeration. Not recommended.
DSheet	22" x 22"
Envelope9	Envelope No. 9; 8.88" x 3.88"
Envelope10	Envelope No. 10; 9.5" x 4.13"
Envelope11	Envelope No. 11; 10.38" x 4.5"
Envelope12	Envelope No. 12; 11" x 4.75"
Envelope14	Envelope No. 14; 11.5" x 5"
EnvelopeB4	ISO B4; 9.84" x 13.9"
EnvelopeB5	ISO B5; 6.9" x 9.8"
EnvelopeB6	ISO B6 (Landscape orientation); 6.93" x 4.92"
EnvelopeC3	ISO C3; 12.76" x 18.03"
EnvelopeC4	ISO C4; 9.01" x 12.76"
EnvelopeC5	ISO C5; 6.38" x 9.01"
EnvelopeC6	ISO C6; 4.49" x 6.38"
EnvelopeC65	ISO C5 width, ISO C6 height; 4.49" x 9.01"
EnvelopeDL	Envelope DL; 4.33" x 8.67"
EnvelopeItaly	Italy envelope size; 4.33" x 9.05"
EnvelopeMonarch	Monarch envelope size; 3.87" x 7.5"
EnvelopePersonal	Personal envelope size; 3.63" x 6.5"
ESheet	22" x 22"
Executive	Executive; 7.25" x 10.5"
FanfoldLegalGerman	Mislabeled - Folio; 8.5" x 13"
FanfoldStdGerman	There are known issues with this enumeration. Not recommended.
FanfoldUS	Mislabeled - German Standard Fanfold; 8.5" x 12"
Folio	Folio size; 8.5" x 13"
Ledger	ANSI B Portrait (Ledger); 11" x 17"
Legal	Legal; 8.5" x 14"
Letter	ANSI A (Letter); 8.5" x 11"
LetterSmall	Letter Small; 7.68" x 10.16"
Note	Letter; 8.5" x 11"

Quarto	Quartro size; 8" x 10"
Statement	Statement; 5.5" x 8.5"
Tabloid	ANSI B Landscape (Tabloid); 17" x 11"

Section.PageSize

Description

Sets or returns the size of printed pages.

Signature: C#

```
public PagePaperSize PageSize{ get; set; }
```

Signature: VB.NET

```
Public Property PageSize() As PagePaperSize
```

Examples

C#

```
sect1.PageSize = Section.PagePaperSize.A4;
```

vb.net

```
sect1.PageSize = Section.PagePaperSize.A4
```

Section.PageWidth

Description

Sets or returns an `int` that represents the page width for this section in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int PageWidth{ get; set; }
```

Signature: VB.NET

```
Public Property PageWidth() As Integer
```

Remarks

Page width may not be smaller than .1" (144 twips) or larger than 22" (31680 twips). Values beyond these will be automatically adjusted.

MS Word equivalent: File menu > Page Setup... > Paper tab > Paper size: section > Width: -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return PageWidth
          int pageWidth = oSection.PageWidth;

          //--- Set PageWidth to 8.5 inches
          oSection.PageWidth = 12240;
```

vb.net

```
'--- Return PageWidth
          Dim pageWidth As Integer = oSection.PageWidth

          '--- Set PageWidth to 8.5 inches
          oSection.PageWidth = 12240
```

Section.RestartPageNum

Description

Sets or returns a `boolean` that represents whether or not page numbering will restart at the beginning of this section.

Signature: C#

```
public boolean RestartPageNum{ get; set; }
```

Signature: VB.NET

```
Public Property RestartPageNum() As Boolean
```

Remarks

This is set to false by default (it will continue numbering pages from the previous section).

MS Word equivalent: Insert menu > Page Numbers... > Format... button > Page numbering section > Continue from previous section (false) OR Start at: (true)

Examples

C#

```
//--- Return RestartPageNum
          bool restartPageNum = oSection.RestartPageNum;

          //--- Set RestartPageNum
          oSection.RestartPageNum = true;
```

vb.net

```
'--- Return RestartPageNum
          Dim restartPageNum As Boolean = oSection.RestartPageNum

          '--- Set RestartPageNum
          oSection.RestartPageNum = True
```

Section.RightMargin

Description

Sets or returns an `int` that represents the horizontal distance of the right margin from the right edge of the page for this section. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int RightMargin{ get; set; }
```

Signature: VB.NET

```
Public Property RightMargin() As Integer
```

Remarks

MS Word equivalent: File menu > Page Setup... > Margins tab > Right: -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return RightMargin
          int rightMargin = oSection.RightMargin;

          //--- Set RightMargin to 1 inch
          oSection.RightMargin = 1440;
```

vb.net

```
'--- Return RightMargin
          Dim rightMargin As Integer = oSection.RightMargin

          '--- Set RightMargin to 1 inch
          oSection.RightMargin = 1440
```

Section.TopMargin

Description

Sets or returns an `int` that represents the vertical distance of the top margin from the top edge of the page for this section. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int TopMargin{ get; set; }
```

Signature: VB.NET

```
Public Property TopMargin() As Integer
```

Remarks

MS Word equivalent: File menu > Page Setup... > Margins tab > Top: -- (Make sure Apply to: This section is selected)

Examples

C#

```
//--- Return TopMargin
          int topMargin = oSection.TopMargin;

          //--- Set TopMargin to 1 inch
          oSection.TopMargin = 1440;
```

vb.net

```
'--- Return TopMargin
          Dim topMargin As Integer = oSection.TopMargin

          '--- Set TopMargin to 1 inch
          oSection.TopMargin = 1440
```

Section.Unlocked

Description

Sets or returns a `boolean` that represents whether or not this section is unlocked.

Signature: C#

```
public boolean Unlocked{ get; set; }
```

Signature: VB.NET

```
Public Property Unlocked() As Boolean
```

Examples

C#

```
//--- Return Unlocked
          bool unlocked = oSection.Unlocked;

          //--- Set Unlocked
          oSection.Unlocked = false;
```

vb.net

```
'--- Return Unlocked
          Dim unlocked As Boolean = oSection.Unlocked

          '--- Set Unlocked
          oSection.Unlocked = False
```

Shading

Description

This class is represents background shading for text, table cells, etc.

Signature: C#

```
public sealed class Shading
```

Signature: VB.NET

```
Public NotInheritable Class Shading
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Shading tab

The following example demonstrates how to get the shading object of a paragraph.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Paragraph paragraph = doc.InsertParagraphAfter(null);
          Shading shad = paragraph.Formatting.Shading;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim paragraph As Paragraph = doc.InsertParagraphAfter(Nothing)
          Dim shad As Shading = paragraph.Formatting.Shading
```

Properties

Name	Description
BackgroundColor	Sets or returns a `Color` object that represents the background (fill) color. This is the main color used for shading.
ForegroundColor	Sets or returns a `Color` object that represents the foreground color. This is the alternate color that is used with patterns.
Pattern	Sets or returns a Shading.ShadingPattern object that represents the shading pattern used.

Nested Classes

Name	Description
ShadingPattern	Shading patterns that can be applied to an element.

Shading.BackgroundColor

Description

Sets or returns a `Color` object that represents the background (fill) color. This is the main color used for shading.

Signature: C#

```
public System.Drawing.Color BackgroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property BackgroundColor() As System.Drawing.Color
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Shading tab > Fill section

Shading.ForegroundColor

Description

Sets or returns a `Color` object that represents the foreground color. This is the alternate color that is used with patterns.

Signature: C#

```
public System.Drawing.Color ForegroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property ForegroundColor() As System.Drawing.Color
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Shading tab > Patterns section > Color:

Examples

C#

```
//--- Return ForegroundColor
          Color clr = shad.ForegroundColor;

          //--- Set ForegroundColor
          shad.ForegroundColor = Color.Yellow;
```

vb.net

```
'--- Return ForegroundColor
          Dim clr As Color = shad.ForegroundColor

          '--- Set ForegroundColor
          shad.ForegroundColor = Color.Yellow
```

Shading.Pattern

Description

Sets or returns a Shading.ShadingPattern object that represents the shading pattern used.

Signature: C#

```
public ShadingPattern Pattern{ get; set; }
```

Signature: VB.NET

```
Public Property Pattern() As ShadingPattern
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Shading tab > Patterns section > Style:

Examples

C#

```
//--- Return Pattern
          ShadingShadingPattern shadPattern = shad.Pattern;

          //--- Set Pattern
          shad.Pattern = Shading.ShadingPattern.DarkVertical;
```

vb.net

```
'--- Return Pattern
          Dim shadPattern As Shading.ShadingPattern = shad.Pattern

          '--- Set Pattern
          shad.Pattern = Shading.ShadingPattern.DarkVertical
```

Shading.ShadingPattern

Description

Shading patterns that can be applied to an element.

Signature: C#

```
public sealed class ShadingPattern : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ShadingPattern
  Inherits TypesafeEnum
```

Values

Name	Description
Auto	Defaults to normal background shading
BackwardDiagonal	Backward diagonal lines
Cross	Crossing vertical and horizontal lines
DarkBackwardDiagonal	Dark Backward diagonal lines
DarkCross	Dark crossing vertical and horizontal lines
DarkDiagonalCross	Dark diagonal crossing lines
DarkForwardDiagonal	Dark forward diagonal lines
DarkHorizontal	Dark horizontal lines
DarkVertical	Dark vertical lines
DiagonalCross	Crossing diagonal lines
ForwardDiagonal	Forward diagonal lines
Horizontal	Horizontal lines
Percent2half	2.5% background color saturation
Percent5	5% background color saturation
Percent7half	7.5% background color saturation
Percent10	10% background color saturation
Percent12half	12.5% background color saturation
Percent15	15% background color saturation
Percent17half	17.5% background color saturation
Percent20	20% background color saturation
Percent22half	22.5% background color saturation

Percent25	25% background color saturation
Percent27half	27.5% background color saturation
Percent30	30% background color saturation
Percent32half	32.5% background color saturation
Percent35	35% background color saturation
Percent37half	37.5% background color saturation
Percent40	40% background color saturation
Percent42half	42.5% background color saturation
Percent45	45% background color saturation
Percent47half	47.5% background color saturation
Percent50	50% background color saturation
Percent52half	52.5% background color saturation
Percent55	55% background color saturation
Percent57half	57.5% background color saturation
Percent60	60% background color saturation
Percent62half	62.5% background color saturation
Percent65	65% background color saturation
Percent67half	67.5% background color saturation
Percent70	70% background color saturation
Percent72half	72.5% background color saturation
Percent75	75% background color saturation
Percent77half	77.5% background color saturation
Percent80	80% background color saturation
Percent82half	82.5% background color saturation
Percent85	85% background color saturation
Percent87half	87.5% background color saturation
Percent90	90% background color saturation
Percent92half	92.5% background color saturation
Percent95	95% background color saturation
Percent97half	97.5% background color saturation
Percent97	dotted background
Solid	Solid background
Vertical	Vertical lines

Styles

Description

This class is used to store the styles of a document. It is essentially a collection of named styles.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Styles
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Styles
```

Remarks

You cannot create styles programmatically. You can only access Named Styles from an existing document. At this time, styles are read-only, but in the future you will be able to modify Named Styles using WordWriter.

The following example demonstrates how to get the styles from a document.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Styles styls = doc.Styles;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim styls As Styles = doc.Styles
```

Indexers

Name	Description
Item(Int32)	Returns a NamedStyle object that represents the named style that has the specified ID.
Item(NamedStyle.BuiltIn)	Returns a NamedStyle object that represents the named style that has the specified built in name.
Item(String)	Returns a NamedStyle object that represents the named style that has the specified string version of its name.

Methods

Name	Description
Length()	Returns an `int` that represents the number of styles in this collection.

Styles.Item

Overload	Description
Item(Int32)	The page Style.Item(Int32) does not exist.
Item(NamedStyle.BuiltIn)	Returns a NamedStyle object that represents the named style that has the specified built in name.
Item(String)	Returns a NamedStyle object that represents the named style that has the specified string version of its name.

Styles.Item(Int32)

Description

Returns a NamedStyle object that represents the named style that has the specified ID.

Signature: C#

```
public NamedStyle this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As NamedStyle
```

Parameters

name

An `int` that represents the ID of the style to retrieve.

Returns

A `NamedStyle` object that represents the named style that has the specified index.

Examples

C#

```
//--- Get Style by name
          NamedStyle myStyle = styls["MyStyle"];

          //--- Get Style by ID
          NamedStyle styleZero = styls[0];
```

vb.net

```
'--- Get Style by name
          Dim myStyle As NamedStyle = styls.Item("MyStyle")

          '--- Get Style by ID
          Dim styleZero As NamedStyle = styls.Item(0)
```

Styles.Item(NamedStyle.BuiltIn)

Description

Returns a NamedStyle object that represents the named style that has the specified built in name.

Signature: C#

```
public NamedStyle this[BuiltIn name] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal name As BuiltIn) As NamedStyle
```

Parameters

name

A NamedStyle.BuiltIn enumeration that represents the name of the style to retrieve.

Returns

A `NamedStyle` object that represents the named style that has the specified built in name.

Examples

C#

```
//--- Get Style by name
          NamedStyle myStyle = styls[NamedStyle.BuiltIn.Normal];
```

vb.net

```
'--- Get Style by name
          Dim myStyle As NamedStyle = styls.Item(NamedStyle.BuiltIn.Normal)
```

Styles.Item(String)

Description

Returns a NamedStyle object that represents the named style that has the specified string version of its name.

Signature: C#

```
public NamedStyle this[System.String name] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal name As String) As NamedStyle
```

Parameters

name

An `String` that represents the name of the style's name to retrieve.

Returns

A `NamedStyle` object that represents the named style that has the specified string version of its name or the specified index.

Examples

C#

```
//--- Get Style by name
          NamedStyle myStyle = styls["MyStyle"];

          //--- Get Style by ID
          NamedStyle styleZero = styls[0];
```

vb.net

```
'--- Get Style by name
          Dim myStyle As NamedStyle = styls.Item("MyStyle")

          '--- Get Style by ID
          Dim styleZero As NamedStyle = styls.Item(0)
```

Styles.Length()

Description

Returns an `int` that represents the number of styles in this collection.

Signature: C#

```
public int Length()
```

Signature: VB.NET

```
Public Function Length() As Integer
```

Examples

C#

```
int length = styls.Length;
```

vb.net

```
Dim length As Integer = styls.Length
```

Table

Description

Represents a table in the Word document.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Table : Element
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Table
  Inherits Element
```

Remarks

To create a new table, use Element.InsertTableBefore() or InsertTableAfter(). To get an existing table, use Element.GetElements or Element.Children and check which elements are of type Element.Type.Table.

The following sample demonstrates inserting a new table at the end of a document as well as finding the first table in a document.

Examples

C#

```
//--- Insert a table at the end of a new document with 3 rows and 5 columns
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          Table table = doc.InsertTableAfter(3, 5);

          //--- Get the first table of an existing document
          WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Table table = doc.GetElements(Element.Type.Table)[0];
```

vb.net

```
'--- Insert a table at the end of a new document with 3 rows and 5 columns
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim table As Table = doc.InsertTableAfter(3, 5)

          '--- Get the first table of an existing document
          Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim table As Table = doc.GetElements(Element.Type.Table)(0)
```

Properties

Name	Description
AbsolutePositioning	Returns an AbsolutePositioning object on which you can control the absolute positioning for the table.
Formatting	Sets or returns a TableFormatting object representing the formatting of this table (borders, padding, shading, etc).
NumRows	Returns an `int` representing the number of rows in the first row of this table.
PreferredWidth	Returns or sets an `int` that represents the preferred width of this table in twips. One twip = (1/20 pt) or (1/1440 in)
Style	Sets or returns a NamedStyle object that represents the Style of the current table.

Indexers

Name	Description
Item(Int32, Int32)	Returns a TableCell object that represents the table cell that exists at the given column and row.

Methods

Name	Description
AddColumns(Int32)	Adds a specified number of columns to the horizontal end of the table.
AddRows(Int32)	Adds a specified number of rows to the vertical end of the table.
CreateBookmarkOnRow(Int32, String)	Creates a bookmark on the specified row using the specified name.
DeleteColumns(Int32, Int32)	{excerpt-include:Table.DeleteColumns(Int32,
DeleteRows(Int32, Int32)	Deletes one or more rows from the table.
GetNumColumns(Int32)	Returns the number of columns in the specified row of the table.
GetPreferredColumnWidth(Int32)	Returns an `int` representing the width of a particular column. The GetPreferredColumnWidth method returns the width of the column in twips. One twip = (1/20 pt) or (1/1440 in).
GetPreferredColumnWidthUnits(Int32)	Returns a TableCell.WidthUnits object that represents the units used for the width of a particular column.
GetPreferredWidth(out Units)	Returns a double that represents the preferred width of this cell, with the associated units return in units.
GetRowHeight(Int32)	Returns an `int` representing the height of a particular row in the table. The default row height unit is twips. One twip = (1/20 pt) or (1/1440 in).
GetRowHeightExact(Int32)	Returns or a `boolean` that represents if the row height is "exact" (true) or "at least" (false).
ImportDataRow(Object())	Imports data from an array of objects to the current table.
InsertColumns(Int32, Int32)	Inserts one or more columns at the specified position in the table.
InsertRows(Int32, Int32)	Inserts one or more rows at the specified position in the table.

MergeCells(Int32, Int32, Int32, Int32)	Merges the specified set of cells into a single cell.
SetPreferredColumnWidth(Int32, Int32)	Sets an `int` representing the width of a particular column. In order to set the width of a column, you must first set the PreferredColumnWidthUnit of that column.
SetPreferredColumnWidthUnits(Int32, TableCell.WidthUnits)	Sets a TableCell.WidthUnits object that represents the units used for the width of a particular column.
SetPreferredWidth(Double, Units)	Sets the preferred width of the table and the units for said width.
SetRowHeight(Int32, Int32)	Sets an `int` representing the height of a particular row in the table. A row height is set in twips. One twip = (1/20 pt) or (1/1440 in).
SetRowHeightExact(Int32, Boolean)	Sets a `boolean` that represents if the row height is "exact" (true) or "at least" (false).

Table.AbsolutePositioning

Description

Returns an AbsolutePositioning object on which you can control the absolute positioning for the table.

Signature: C#

```
public AbsolutePositioning AbsolutePositioning{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property AbsolutePositioning() As AbsolutePositioning
```

Examples

C#

```
oTable.AbsolutePositioning.X = 100;
```

vb.net

```
oTable.AbsolutePositioning.X = 100
```

Table.AddColumns(Int32)

Description

Adds a specified number of columns to the horizontal end of the table.

Signature: C#

```
public void AddColumns(int numColumns)
```

Signature: VB.NET

```
Public Sub AddColumns(ByVal numColumns As Integer)
```

Parameters

numColumns

The number of columns to add to the table.

Examples

C#

```
tbl.AddColumns(4);
```

vb.net

```
tbl.AddColumns(4)
```

Table.AddRows(Int32)

Description

Adds a specified number of rows to the vertical end of the table.

Signature: C#

```
public void AddRows(int numRows)
```

Signature: VB.NET

```
Public Sub AddRows(ByVal numRows As Integer)
```

Parameters

numRows

The number of rows to add to the table.

Examples

C#

```
tbl.AddRows(4);
```

vb.net

```
tbl.AddRows(4)
```

Table.CreateBookmarkOnRow(Int32, String)

Description

Creates a bookmark on the specified row using the specified name.

Signature: C#

```
public void CreateBookmarkOnRow(int row, System.String name)
```

Signature: VB.NET

```
Public Sub CreateBookmarkOnRow(ByVal row As Integer, ByVal name As String)
```

Parameters

row

An `int` representing the table cell's row.

name

A `String` representing the bookmark name.

Examples

C#

```
oTable.CreateBookmarkOnRow(1, "MyBookmark");
```

vb.net

```
oTable.CreateBookmarkOnRow(1, "MyBookmark")
```

Table.DeleteColumns(Int32, Int32)

Description

Deletes one or more columns from the table.

Signature: C#

```
public void DeleteColumns(int startIndex, int numColumns)
```

Signature: VB.NET

```
Public Sub DeleteColumns(ByVal startIndex As Integer, ByVal numColumns As Integer)
```

Parameters

startIndex

The 0-based index of the first column to delete.

numColumns

The number of columns to delete.

Examples

C#

```
tbl.DeleteColumns(2, 5);
```

vb.net

```
tbl.DeleteColumns(2, 5)
```

Table.DeleteRows(Int32, Int32)

Description

Deletes one or more rows from the table.

Signature: C#

```
public void DeleteRows(int startIndex, int numRows)
```

Signature: VB.NET

```
Public Sub DeleteRows(ByVal startIndex As Integer, ByVal numRows As Integer)
```

Parameters

startIndex

The 0-based index of the first row to delete.

numRows

The number of rows to delete.

Examples

C#

```
tbl.DeleteRows(2, 5);
```

vb.net

```
tbl.DeleteRows(2, 5)
```

Table.Formatting

Description

Sets or returns a TableFormatting object representing the formatting of this table (borders, padding, shading, etc).

Signature: C#

```
public TableFormatting Formatting{ get; set; }
```

Signature: VB.NET

```
Public Property Formatting() As TableFormatting
```

Examples

C#

```
//--- Get TableFormatting
          TableFormatting formatting = oTable.TableFormatting;

          //--- Set TableFormatting
          oTable.TableFormatting = oTableFormatting;
```

vb.net

```
'--- Get TableFormatting
          Dim formatting As TableFormatting = oTable.TableFormatting

          '--- Set TableFormatting
          oTable.TableFormatting = oTableFormatting
```

Table.GetNumColumns(Int32)

Description

Returns the number of columns in the specified row of the table.

Signature: C#

```
public int GetNumColumns(int row)
```

Signature: VB.NET

```
Public Function GetNumColumns(ByVal row As Integer) As Integer
```

Parameters

row

The row for which to return the nummber of columns.

Examples

C#

```
tbl.GetNumColumns(4);
```

vb.net

```
tbl.GetNumColumns(4)
```

Table.GetPreferredColumnWidth(Int32)

Description

Returns an `int` representing the width of a particular column. The GetPreferredColumnWidth method returns the width of the column in twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int GetPreferredColumnWidth(int column)
```

Signature: VB.NET

```
Public Function GetPreferredColumnWidth(ByVal column As Integer) As Integer
```

Parameters

column

An `int` representing the column index of the desired column.

Returns

An `int` representing the width in twips for the column at the specified column index.

Remarks

MS Word equivalent: Table menu > Table properties... > Column tab > Size section > Column #: > Preferred width:

Examples

C#

```
//--- Get the width of the 3rd column
          int columnWidth = oTable.GetPreferredColumnWidth(2);
```

vb.net

```
'--- Get the width of the 3rd column
          Dim columnWidth As Integer = oTable.GetPreferredColumnWidth(2)
```

Table.GetPreferredColumnWidthUnits(Int32)

Description

Returns a TableCell.WidthUnits object that represents the units used for the width of a particular column.

Signature: C#

```
public WidthUnits GetPreferredColumnWidthUnits(int column)
```

Signature: VB.NET

```
Public Function GetPreferredColumnWidthUnits(ByVal column As Integer) As WidthUnits
```

Parameters

column

An `int` representing the column index for which to get the width units.

Returns

A `TableCell.WidthUnits` object representing the units used at the specified column index.

Remarks

MS Word equivalent: Table menu > Table properties... > Column tab > Size section > Column #: > Measure in:

Examples

C#

```
//--- Get the column width units for the 3rd column
          TableCell.WidthUnits oCellWidthUnits =
               oTable.GetPreferredColumnWidthUnits(2);
```

vb.net

```
'--- Get the column width units for the 3rd column
          Dim oCellWidthUnits As TableCell.WidthUnits = _
               oTable.GetPreferredColumnWidthUnits(2)
```

Table.GetPreferredWidth(out Units)

Introduced in build 4.1.0.1331

Description

Returns a double that represents the preferred width of this cell, with the associated units return in units.

Signature: C#

```
public double GetPreferredWidth(out Units units)
```

Signature: VB.NET

```
Public Function GetPreferredWidth(ByRef units As Units) As Double
```

Parameters

units

The units for the width.
MS Word equivalent: Table menu > Table Properties... > Table tab > Options... > Measure in

Returns

The preferred width of the table.
MS Word equivalent: Table menu > Table Properties... > Table tab > Options... > dropdown after "Preferred width"

Examples

C#

```
Units units;
double width = table.GetPreferredWidth(out units);
```

vb.net

```
Dim units As Units
Dim width As Double = table.GetPreferredWidth(units)
```

Table.GetRowHeight(Int32)

Description

Returns an `int` representing the height of a particular row in the table. The default row height unit is twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int GetRowHeight(int row)
```

Signature: VB.NET

```
Public Function GetRowHeight(ByVal row As Integer) As Integer
```

Parameters

row

An `int` representing the index of the desired row.

Returns

An `int` representing the height in twips of the specified row. One twip = (1/20 pt) or (1/1440 in).

Remarks

MS Word equivalent: Table menu > Table properties... > Row tab > Size section > Row #: > Specify height:

Examples

C#

```
//--- Get the height of the 3rd row
          int rowHeight = oTable.GetRowHeight(2);
```

vb.net

```
'--- Get the height of the 3rd row
          Dim rowHeight As Integer = oTable.GetRowHeight(2)
```

Table.GetRowHeightExact(Int32)

Description

Returns or a `boolean` that represents if the row height is "exact" (true) or "at least" (false).

Signature: C#

```
public boolean GetRowHeightExact(int row)
```

Signature: VB.NET

```
Public Function GetRowHeightExact(ByVal row As Integer) As Boolean
```

Parameters

row

An `int` representing the row index to get RowHeightExact for.

Returns

A `boolean` that represents if the specified row's height is "exact" (true) or "at least" (false).

Remarks

MS Word equivalent: Table menu > Table properties... > Row tab > Size section > Row #: > Row height is:

Note: To set this property, you must first set the RowHeight property.

Examples

C#

```
//--- Get RowHeightExact for the 3rd row
          bool rowHeightExact = oTable.GetRowHeightExact(2);
```

vb.net

```
'--- Get RowHeightExact for the 3rd row
          Dim rowHeightExact As Boolean = oTable.GetRowHeightExact(2)
```

Table.ImportDataRow(Object())

Description

Imports data from an array of objects to the current table.

Signature: C#

```
public void ImportDataRow(System.Object[] row)
```

Signature: VB.NET

```
Public Sub ImportDataRow(ByVal row As Object())
```

Parameters

row

An array containing the set of values to import to the table.

Exceptions

System.Exception

Examples

C#

```
tbl.ImportData(dataArray);
```

vb.net

```
tbl.ImportData(dataArray)
```

Table.InsertColumns(Int32, Int32)

Description

Inserts one or more columns at the specified position in the table.

Signature: C#

```
public void InsertColumns(int index, int numColumns)
```

Signature: VB.NET

```
Public Sub InsertColumns(ByVal index As Integer, ByVal numColumns As Integer)
```

Parameters

index

The 0-based index of the first column to insert.

numColumns

The number of columns to insert.

Examples

C#

```
tbl.InsertColumns(2, 5);
```

vb.net

```
tbl.InsertColumns(2, 5)
```

Table.InsertRows(Int32, Int32)

Description

Inserts one or more rows at the specified position in the table.

Signature: C#

```
public void InsertRows(int index, int numRows)
```

Signature: VB.NET

```
Public Sub InsertRows(ByVal index As Integer, ByVal numRows As Integer)
```

Parameters

index

The 0-based index of the first row to insert.

numRows

The number of rows to insert.

Examples

C#
`tbl.InsertRows(2, 5);`

vb.net
`tbl.InsertRows(2, 5)`

Table.Item(Int32, Int32)

Description

Returns a TableCell object that represents the table cell that exists at the given column and row.

Signature: C#

```
public TableCell this[int row, int column] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal row As Integer, ByVal column As Integer) As TableCell
```

Parameters

row

An `int` representing the table cell's row.

column

An `int` representing the table cell's column.

Returns

An `TableCell` object that represents the table cell that exists at the given column and row.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Table table = doc.InsertTableAfter(1, 1);
          TableCell cell = table[0, 0];
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim tbl As Table = doc.InsertTableAfter(1, 1)
          Dim tc As TableCell = tbl(0, 0)
```

Table.MergeCells(Int32, Int32, Int32, Int32)

Description

Merges the specified set of cells into a single cell.

Signature: C#

```
public void MergeCells(int firstRow, int firstCol, int numRows, int numCols)
```

Signature: VB.NET

```
Public Sub MergeCells(ByVal firstRow As Integer, ByVal firstCol As Integer, ByVal numRows As Integer,
ByVal numCols As Integer)
```

Parameters

firstRow

The 0-based number of the first row in the set of cells to merge.

firstCol

The 0-based number of the first column in the set of cells to merge.

numRows

The number of rows to merge.

numCols

The number of columns to merge.

Examples

C#

```
table.MergeCells(3, 2, 4, 1);
```

vb.net

```
table.MergeCells(3, 2, 4, 1)
```

Table.NumRows

Description

Returns an `int` representing the number of rows in the first row of this table.

Signature: C#

```
public int NumRows{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NumRows() As Integer
```

Examples

C#

```
int numRows = oTable.NumRows;
```

vb.net

```
Dim numRows As Integer = oTable.NumRows
```

Table.PreferredWidth

Since build 4.1.0.1331: Use SetPreferredWidth(Double, Units) and GetPreferredWidth(out Units) instead

Description

Returns or sets an `int` that represents the preferred width of this table in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
[Obsolete("Use SetPreferredWidth(Double,Units) instead")]
 public int PreferredWidth{ get; set; }
```

Signature: VB.NET

```
<Obsolete("Use SetPreferredWidth(Double,Units) instead")> _
Public Property PreferredWidth() As Integer
```

Remarks

By default, this value is not set for a new table.

Word equivalent: Table menu > Table Properties... > Table tab > Size section > Preferred width:

Examples

C#

```
//--- Return PreferredWidth
          int preferredWidth = oTable.PreferredWidth;

          //--- Set PreferredWidth to 6 inches
          oTable.PreferredWidth = 8640;
```

vb.net

```
'--- Return PreferredWidth
          Dim repeatAsHeader As Integer = oTable.PreferredWidth

          '--- Set PreferredWidth to 6 inches
          oTable.PreferredWidth = 8640
```

Table.SetPreferredColumnWidth(Int32, Int32)

Description

Sets an `int` representing the width of a particular column. In order to set the width of a column, you must first set the PreferredColumnWidthUnit of that column.

Signature: C#

```
public void SetPreferredColumnWidth(int column, int width)
```

Signature: VB.NET

```
Public Sub SetPreferredColumnWidth(ByVal column As Integer, ByVal width As Integer)
```

Parameters

column

An `int` representing the index of the column for which to change the width.

width

An `int` representing the column's new width in twips. One twip = (1/20 pt) or (1/1440 in).

Remarks

MS Word equivalent: Table menu > Table properties... > Column tab > Size section > Column #: > Preferred width:

Examples

C#

```
//--- Set the width of the 3rd column to 1 inch
          oTable.SetPreferredColumnWidthUnits(2, TableCell.WidthUnits.Twips);
          oTable.SetPreferredColumnWidth(2, 1440);
```

vb.net

```
'--- Set the width of the 3rd column to 1 inch
          oTable.SetPreferredColumnWidthUnits(2, TableCell.WidthUnits.Twips)
          oTable.SetPreferredColumnWidth(2, 1440)
```

Table.SetPreferredColumnWidthUnits(Int32, TableCell.WidthUnits)

Description

Sets a TableCell.WidthUnits object that represents the units used for the width of a particular column.

Signature: C#

```
public void SetPreferredColumnWidthUnits(int column, WidthUnits units)
```

Signature: VB.NET

```
Public Sub SetPreferredColumnWidthUnits(ByVal column As Integer, ByVal units As WidthUnits)
```

Parameters

column

The column index for which to get the width units.

units

The preferred width units.

Remarks

MS Word equivalent: Table menu > Table properties... > Column tab > Size section > Column #: > Measure in:

Examples

C#

```
//--- Set the column width units for the 3rd column to percentage
          oTable.SetPreferredWidthUnits(2, TableCell.WidthUnits.Percentage);
```

vb.net

```
'--- Set the column width units for the 3rd column to percentage
          oTable.SetPreferredWidthUnits(2, TableCell.WidthUnits.Percentage)
```

Table.SetPreferredWidth(Double, Units)

Introduced in build 4.1.0.1331

Description

Sets the preferred width of the table and the units for said width.

Signature: C#

```
public void SetPreferredWidth(double width, Units units)
```

Signature: VB.NET

```
Public Sub SetPreferredWidth(ByVal width As Double, ByVal units As Units)
```

Parameters

width

The preferred width of the table.
MS Word equivalent: Table menu > Table Properties... > Table tab > Options... > dropdown after "Preferred width"

units

The units for the width.
For Units.Auto:
MS Word equivalent: Table menu > Table Properties... > Table tab > Options... > Preferred width
For Units.Percent, or Units.Twips:
MS Word equivalent: Table menu > Table Properties... > Table tab > Options... > Measure in

Exceptions

ArgumentOutOfRangeException

If units is something other than Units.Auto,Units.Percent, or Units.Twips.

Remarks

Examples

C#

```
// 50%
table.SetPreferredWidth(50, Units.Percent);
// or 2"
table.SetPreferredWidth(2880, Units.Twips);
// or auto fit
table.SetPreferredWidth(0,Units.Auto);
```

vb.net

```
' 50%
table.SetPreferredWidth(50, Units.Percent)
' or 2"
table.SetPreferredWidth(2880, Units.Twips)
' or auto fit
table.SetPreferredWidth(0, Units.Auto)
```

Table.SetRowHeight(Int32, Int32)

Description

Sets an `int` representing the height of a particular row in the table. A row height is set in twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public void SetRowHeight(int row, int height)
```

Signature: VB.NET

```
Public Sub SetRowHeight(ByVal row As Integer, ByVal height As Integer)
```

Parameters

row

An `int` representing the row index to set the height for.

height

An `int` representing the new height in twips of the specified row. One twip = (1/20 pt) or (1/1440 in).

Remarks

MS Word equivalent: Table menu > Table properties... > Row tab > Size section > Row #: > Specify height:

Examples

C#

```
//--- Set the height of the 3rd row to 1 inch
          oTable.SetRowHeight(2, 1440);
```

vb.net

```
'--- Set the height of the 3rd row to 1 inch
          oTable.SetRowHeight(2, 1440)
```

Table.SetRowHeightExact(Int32, Boolean)

Description

Sets a `boolean` that represents if the row height is "exact" (true) or "at least" (false).

Signature: C#

```
public void SetRowHeightExact(int row, boolean exact)
```

Signature: VB.NET

```
Public Sub SetRowHeightExact(ByVal row As Integer, ByVal exact As Boolean)
```

Parameters

row

An `int` representing the row index to set RowHeightExact for.

exact

A `boolean` that represents if the specified row's height should be "exact" (true) or "at least" (false).

Remarks

MS Word equivalent: Table menu > Table properties... > Row tab > Size section > Row #: > Row height is:

Note: To set this property, you must first set the RowHeight property.

Examples

C#

```
//--- Set RowHeightExact for the 3rd row to exact
          oTable.SetRowHeightExact(2, true);
```

vb.net

```
'--- Set RowHeightExact for the 3rd row to exact
          oTable.SetRowHeightExact(2, True)
```

Table.Style

Description

Sets or returns a NamedStyle object that represents the Style of the current table.

Signature: C#

```
public NamedStyle Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As NamedStyle
```

Examples

C#

```
//--- Return NamedStyle
          NamedStyle oNamedStyle = oTable.Style;

          //--- Set NamedStyle
          oTable.Style = oDocument.Styles[NamedStyle.BuiltIn.NormalTable];
```

vb.net

```
'--- Return NamedStyle
          Dim oNamedStyle As NamedStyle = oTable.Style

          '--- Set NamedStyle
          oTable.Style = oDocument.Styles(NamedStyle.BuiltIn.NormalTable)
```

TableCell

Description

Represents a table cell in a table in the Word document.

Signature: C#

```
public sealed class TableCell : Element
```

Signature: VB.NET

```
Public NotInheritable Class TableCell
  Inherits Element
```

Remarks

New table cells are created when you create a new Table. They cannot be created independent of a Table using the `new` keyword.

To retrieve and manipulate a table cell, use array index notation on a `Table` object. You can also use Element.GetElements(Element.Type.TableCell) to get all elements, of which you can check which ones are of type TableCell.

The following sample demonstrates getting a reference to a table cell at column 3 (4th column), row 2 (3rd row).

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          Table table = (Table)doc.GetElements(Element.Type.Table)[0];
          TableCell cell = table[3, 2];
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim table As Table = doc.GetElements(Element.Type.Table)(0)
          Dim cell As TableCell = table(3, 2)
```

Properties

Name	Description
CellWidthUnits	Sets or returns a TableCell.WidthUnits object that represents the cell width units for this cell.
FirstMerged	Returns or sets a `boolean` that represents if this is the first cell in a group of horizontally merged cells. The first cell will contain all the content.
FirstVerticalMerged	Sets or returns whether the current cell is the first cell in a group of vertically merged cells. The first cell will contain all the content.
Merged	Sets or returns whether this cell has been merged with a preceding horizontal cell.
NoWrap	Returns or sets a `boolean` that represents if Word will prevent text from wrapping in this table cell.
PreferredWidth	Returns or sets an `int` that represents the preferred width of this cell. In order to set this property, the CellWidthUnits property must first be set. The width is returned in twips. One twip = (1/20 pt) or (1/1440 in).
Shading	Returns an Shading object which on which you can manipulate the shading (fill color and/or pattern) properties for this cell.
VertAlignment	Returns or sets a TableCell.VerticalAlignment object that represents the vertical alignment of the text in this cell.
VerticalMerged	Returns or sets a `boolean` that represents if this cell is merged with one or more cells vertically. The first cell will contain all the content.

Methods

Name	Description
ApplyFont(Font)	Applies Font properties to all text in this TableCell. The font properties will not be applied to text inside nested tables.
ApplyParagraphFormatting(ParagraphFormatting)	Applies ParagraphFormatting to all paragraphs in this TableCell. The ParagraphFormatting will not be applied to paragraphs inside nested tables.
GetBorder(Border.Location)	Returns a Border object that lets you manipulate the border this table cell at a particular location.
GetPadding(TableCell.Location)	Returns an `int` that represents this table cell margin at a particular location in twips. One twip = (1/20 pt) or (1/1440 in)
SetPadding(TableCell.Location, Int32)	Sets an `int` that represents this table cell margin at a particular location in twips. One twip = (1/20 pt) or (1/1440 in)

Nested Classes

Name	Description
Location	Locations within a table cell for which padding values can be set.
VerticalAlignment	Vertical alignments for a table cell.
WidthUnits	Units used for setting a table cell width.

TableCell.ApplyFont(Font)

Description

Applies Font properties to all text in this TableCell. The font properties will not be applied to text inside nested tables.

Signature: C#

```
public void ApplyFont(Font font)
```

Signature: VB.NET

```
Public Sub ApplyFont(ByVal font As Font)
```

Parameters

font

A `Font` representing the properties that should be applied to text in this TableCell.

Examples

C#

```
tblCell.ApplyFont(tblFont);
```

vb.net

```
tblCell.ApplyFont(tblFont)
```

TableCell.ApplyParagraphFormatting(ParagraphFormatting)

Description

Applies ParagraphFormatting to all paragraphs in this TableCell. The ParagraphFormatting will not be applied to paragraphs inside nested tables.

Signature: C#

```
public void ApplyParagraphFormatting(ParagraphFormatting formatting)
```

Signature: VB.NET

```
Public Sub ApplyParagraphFormatting(ByVal formatting As ParagraphFormatting)
```

Parameters

formatting

A `ParagraphFormatting` object containing formatting properties to apply to the cell.

Examples

C#

```
tblCell.ApplyParagraphFormatting(paraFormat);
```

vb.net

```
tblCell.ApplyParagraphFormatting(paraFormat)
```

TableCell.CellWidthUnits

Description

Sets or returns a TableCell.WidthUnits object that represents the cell width units for this cell.

Signature: C#

```
public WidthUnits CellWidthUnits{ get; set; }
```

Signature: VB.NET

```
Public Property CellWidthUnits() As WidthUnits
```

Remarks

MS Word equivalent: Table menu > Table Properties... > Cell tab > Size section > Measure in:

Examples

C#

```
//--- Return CellWidthUnits
          TableCell.WidthUnits widthUnits = cell.CellWidthUnits;

          //--- Set CellWidthUnits
          cell.CellWidthUnits = TableCell.WidthUnits.Auto;
```

vb.net

```
'--- Return CellWidthUnits
          Dim widthUnits As TableCell.WidthUnits = cell.CellWidthUnits

          '--- Set CellWidthUnits
          cell.CellWidthUnits = TableCell.WidthUnits.Auto
```

TableCell.FirstMerged

Description

Returns or sets a `boolean` that represents if this is the first cell in a group of horizontally merged cells. The first cell will contain all the content.

Signature: C#

```
public boolean FirstMerged{ get; set; }
```

Signature: VB.NET

```
Public Property FirstMerged() As Boolean
```

Examples

C#

```
//--- Return FirstMerged
          bool merged = cell.FirstMerged;

          //--- Set FirstMerged
          cell.FirstMerged = true;
```

vb.net

```
'--- Return FirstMerged
          Dim merged As Boolean = cell.FirstMerged

          '--- Set FirstMerged
          cell.FirstMerged = True
```

TableCell.FirstVerticalMerged

Description

Sets or returns whether the current cell is the first cell in a group of vertically merged cells. The first cell will contain all the content.

Signature: C#

```
public boolean FirstVerticalMerged{ get; set; }
```

Signature: VB.NET

```
Public Property FirstVerticalMerged() As Boolean
```

Examples

C#

```
//--- Return FirstVerticalMerged
          bool merged = cell.FirstVerticalMerged;

          //--- Set FirstVerticalMerged
          cell.FirstVerticalMerged = true;
```

vb.net

```
'--- Return FirstVerticalMerged
          Dim merged As Boolean = cell.FirstVerticalMerged

          '--- Set FirstVerticalMerged
          cell.FirstVerticalMerged = True
```

TableCell.GetBorder(Border.Location)

Description

Returns a Border object that lets you manipulate the border this table cell at a particular location.

Signature: C#

```
public Border GetBorder(Location location)
```

Signature: VB.NET

```
Public Function GetBorder(ByVal location As Location) As Border
```

Parameters

location

A Location object that represents the location of the Border to return.

Returns

A `Border` object that represents the border for this table cell at a particular location.

Remarks

Allowed locations are Top, Left, Bottom, and Right. They refer to the border for those sides of the table cell.

Word equivalent: Format menu > Borders and Shading... > Borders tab

Examples

C#

```
//--- Set the Left Border color to blue
          cell.GetBorder(Border.Location.Left).Color = Color.Blue;
```

vb.net

```
'--- Set the Left Border color to blue
          cell.GetBorder(Border.Location.Left).Color = Color.Blue
```

TableCell.GetPadding(TableCell.Location)

Description

Returns an `int` that represents this table cell margin at a particular location in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int GetPadding(Location location)
```

Signature: VB.NET

```
Public Function GetPadding(ByVal location As Location) As Integer
```

Parameters

location

A TableCell.Location object that represents ths location of the margin.

Returns

An `int` that represents this table cell margin at a particular location in twips.

Remarks

Allowed locations are Top, Left, Bottom, and Right, which refer to the default margins for those sides of a table cell.

By default, Top and Bottom are set to 0", and Left and Right are set to 0.08".

Word equivalent: Table menu > Table Properties... > Cell tab > Options... button > Cell margins section

Examples

C#

```
//--- Return Padding for Left margin
          int padding = cell.GetPadding(TableCell.Location.Left);
```

vb.net

```
'--- Return Padding for Left margin
          Dim padding As Integer = cell.GetPadding(TableCell.Location.Left)
```

TableCell.Location

Description

Locations within a table cell for which padding values can be set.

Signature: C#

```
public sealed class Location : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Location
  Inherits TypesafeEnum
```

Values

Name	Description
Bottom	Bottom of the table cell
Left	Left side of the table cell
Right	Right side of the table cell
Top	Top of the table cell

TableCell.Merged

Description

Sets or returns whether this cell has been merged with a preceding horizontal cell.

Signature: C#

```
public boolean Merged{ get; set; }
```

Signature: VB.NET

```
Public Property Merged() As Boolean
```

Examples

C#

```
//--- Return Merged
          bool merged = cell.Merged;

          //--- Set Merged
          cell.Merged = true;
```

vb.net

```
'--- Return Merged
          Dim merged As Boolean = cell.Merged

          '--- Set Merged
          cell.Merged = True
```

TableCell.NoWrap

Description

Returns or sets a `boolean` that represents if Word will prevent text from wrapping in this table cell.

Signature: C#

```
public boolean NoWrap{ get; set; }
```

Signature: VB.NET

```
Public Property NoWrap() As Boolean
```

Remarks

This is set to `false` by default.

MS Word equivalent: Table menu > Table Properties... > Cell tab > Options... button > Options section > Wrap text

Examples

C#

```
//--- Return NoWrap
          bool noWrap = cell.NoWrap;

          //--- Set NoWrap
          cell.NoWrap = true;
```

vb.net

```
'--- Return NoWrap
          Dim noWrap As Boolean = cell.NoWrap

          '--- Set NoWrap
          cell.NoWrap = True
```

TableCell.PreferredWidth

Description

Returns or sets an `int` that represents the preferred width of this cell. In order to set this property, the CellWidthUnits property must first be set. The width is returned in twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int PreferredWidth{ get; set; }
```

Signature: VB.NET

```
Public Property PreferredWidth() As Integer
```

Remarks

MS Word equivalent: Table menu > Table Properties... > Cell tab > Size section > Preferred width:

Examples

C#

```
//--- Return PreferredWidth
          int preferredWidth = cell.PreferredWidth;

          //--- Set PreferredWidth to 1 inch
          cell.CellWidthUnits = TableCell.WidthUnits.Twips;
          cell.PreferredWidth = 1440;
```

vb.net

```
'--- Return PreferredWidth
          Dim preferredWidth As Integer = cell.PreferredWidth

          '--- Set PreferredWidth to 1 inch
          cell.CellWidthUnits = TableCell.WidthUnits.Twips
          cell.PreferredWidth = 1440
```

TableCell.SetPadding(TableCell.Location, Int32)

Description

Sets an `int` that represents this table cell margin at a particular location in twips. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public void SetPadding(Location location, int value)
```

Signature: VB.NET

```
Public Sub SetPadding(ByVal location As Location, ByVal value As Integer)
```

Parameters

location

A TableCell.Location object that represents ths location of the margin.

value

An `int` representing the value of the margin in twips.

Remarks

Allowed locations are Top, Left, Bottom, and Right, which refer to the default margins for those sides of a table cell.

By default, Top and Bottom are set to 0", and Left and Right are set to 0.08".

Word equivalent: Table menu > Table Properties... > Cell tab > Options... button > Cell margins section

Examples

C#

```
//--- Set Padding for Left margin to 1 inch
          cell.SetPadding(TableCell.Location.Left, 1440);
```

vb.net

```
'--- Set Padding for Left margin to 1 inch
          cell.SetPadding(TableCell.Location.Left, 1440)
```

TableCell.Shading

Description

Returns an Shading object which on which you can manipulate the shading (fill color and/or pattern) properties for this cell.

Signature: C#

```
public Shading Shading{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Shading() As Shading
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Shading tab

Examples

C#

```
cell.Shading.BackgroundColor = Color.Gray;
```

vb.net

```
cell.Shading.BackgroundColor = Color.Gray
```

TableCell.VertAlignment

Description

Returns or sets a TableCell.VerticalAlignment object that represents the vertical alignment of the text in this cell.

Signature: C#

```
public VerticalAlignment VertAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property VertAlignment() As VerticalAlignment
```

Remarks

This is set to TableCell.VerticalAlignment.Top by default.

MS Word equivalent: Table menu > Table Properties... > Cell tab > Vertical alignment

Examples

C#

```
//--- Return VertAlignment
          TableCell.VerticalAlignment oVerticalAlignment = cell.VertAlignment;

          //--- Set VertAlignment
          cell.VertAlignment = TableCell.VerticalAlignment.Center;
```

vb.net

```
'--- Return VertAlignment
          Dim oVerticalAlignment As TableCell.VerticalAlignment = cell.VertAlignment

          '--- Set VertAlignment
          cell.VertAlignment = TableCell.VerticalAlignment.Center
```

TableCell.VerticalAlignment

Description

Vertical alignments for a table cell.

Signature: C#

```
public sealed class VerticalAlignment : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class VerticalAlignment
  Inherits TypesafeEnum
```

Values

Name	Description
Bottom	Text is aligned from the bottom of the cell
Center	Text is aligned from the center of the cell
Top	Text is aligned from the top of the cell

TableCell.VerticalMerged

Description

Returns or sets a `boolean` that represents if this cell is merged with one or more cells vertically. The first cell will contain all the content.

Signature: C#

```
public boolean VerticalMerged{ get; set; }
```

Signature: VB.NET

```
Public Property VerticalMerged() As Boolean
```

Examples

C#

```
//--- Return VerticalMerged
          bool merged = cell.VerticalMerged;

          //--- Set VerticalMerged
          cell.VerticalMerged = true;
```

vb.net

```
'--- Return VerticalMerged
          Dim merged As Boolean = cell.VerticalMerged

          '--- Set VerticalMerged
          cell.VerticalMerged = True
```

TableCell.WidthUnits

Description

Units used for setting a table cell width.

Signature: C#

```
public sealed class WidthUnits : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class WidthUnits
  Inherits TypesafeEnum
```

Values

Name	Description
Auto	Width units are automatically chosen
Null	Default width unit
Percentage	Cell width is measured as a percentage of total width
Twips	Cell width is measured in twips

TableFormatting

Description

The TableFormatting class is used to specify formatting that should be applied to a table. It also represents the table formatting of some named styles in the document.

Signature: C#

```
public sealed class TableFormatting
```

Signature: VB.NET

```
Public NotInheritable Class TableFormatting
```

Remarks

There are two ways to obtain an instance of this class: Create a new normal table formatting object using Document.CreateTableFormatting(). This will return a copy of a particular style's table formatting. The `TableFormatting` object that is returned can then be used in conjunction with methods in the Element class to create tables with specific formatting.

The following example demonstrates both ways of getting table formatting, first by retrieving the NormalTable style's paragraph formatting from the document, second by retrieving a copy of the TableSimple1 style's paragraph formatting.

Examples

C#

```
//--- Get NormalTable formatting from Document
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          TableFormatting normalTableFormatting =
               doc.CreateTableFormatting();

          //--- Get paragraph formatting from SimpleTable1 Style
          WordApplication app = new WordApplication();
          Document doc = app.Create();
          TableFormatting simpleTable1Formatting =
               doc.Styles[NamedStyle.BuiltIn.SimpleTable1];
```

vb.net

```
'--- Get NormalTable formatting from Document
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim normalTableFormatting As TableFormatting = _
               doc.CreateTableFormatting()

          '--- Get paragraph formatting from SimpleTable1 Style
          Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim simpleTable1Formatting As TableFormatting = _
               doc.Styles(NamedStyle.BuiltIn.SimpleTable1)
```

Properties

Name	Description
AutoWidth	Returns a `boolean` that represents whether or not this table's width will be set automatically by Word.
CantSplit	Sets or returns a `boolean` that represents whether or not Word will attempt to keep a table created with this formatting on one page.
DefaultShading	Returns an Shading object which on which you can manipulate the default shading (fill color and/or pattern) properties of a table's cells that has this formatting.
DefaultSpacing	Returns or sets an `int` that represents the default spacing between a table's cells that has this formatting. This is the distance in twips between the edge of a cell and the center of its border with an adjacent cell. One twip = (1/20 pt) or (1/1440 in)
Justification	Sets or returns a ParagraphFormatting.Justification object that represents the justification for a table created with this formatting. Possible values: Left, Center, Right, LeftRight.
LeftIndent	Sets or returns an `int` that represents the left indent for a table created with this formatting in twips. One twip = (1/20 pt) or (1/1440 in).
RepeatAsHeader	Sets or returns a `boolean` that represents if Word will repeat the first row of a table created with this formatting as a header across multiple pages.

Methods

Name	Description

GetDefaultBorder(Border.Location)	Returns a Border object that represents the default border for table cells at a specified location for a table created with this formatting. Individual table cells can override these border settings. Allowed locations are: Top, Left, Bottom, Right, Vertical, and Horizontal. Top, Left, Bottom, and Right refer to the default border for those sides of a table cell. Vertical and Horizontal refer to the border between table cells.
GetDefaultPadding(TableCell.Location)	Returns an `int` that represents the default table cell margin at a particular location. One twip = (1/20 pt) or (1/1440 in)
SetDefaultPadding(TableCell.Location, Int32)	Sets an `int` that represents the default table cell margin at a particular location. One twip = (1/20 pt) or (1/1440 in)

TableFormatting.AutoWidth

Description

Returns a `boolean` that represents whether or not this table's width will be set automatically by Word.

Signature: C#

```
public boolean AutoWidth{ get; set; }
```

Signature: VB.NET

```
Public Property AutoWidth() As Boolean
```

Remarks

MS Word equivalent: Table menu > Table Properties... > Table tab > Options... > Automatically resize to fit contents

> You will have to call `table.SetPreferredColumnWidthUnits(i,TableCell.WidthUnits.Auto)` to have each column adjusted to fit the text

Examples

C#

```
boolean autoWidth = oTableFormatting.AutoWidth;
```

vb.net

```
Dim autoWidth As Boolean = oTableFormatting.AutoWidth
```

TableFormatting.CantSplit

Description

Sets or returns a `boolean` that represents whether or not Word will attempt to keep a table created with this formatting on one page.

Signature: C#

```
public boolean CantSplit{ get; set; }
```

Signature: VB.NET

```
Public Property CantSplit() As Boolean
```

Examples

C#

```
//--- Return CantSplit
          boolean cantSplit = oTableFormatting.CantSplit;

          //--- Set CantSplit
          oTableFormatting.CantSplit = true;
```

vb.net

```
'--- Return CantSplit
          Dim cantSplit As Boolean = oTableFormatting.CantSplit

          '--- Set CantSplit
          oTableFormatting.CantSplit = True
```

TableFormatting.DefaultShading

Description

Returns an Shading object which on which you can manipulate the default shading (fill color and/or pattern) properties of a table's cells that has this formatting.

Signature: C#

```
public Shading DefaultShading{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DefaultShading() As Shading
```

Remarks

MS Word equivalent: Format menu > Borders and Shading... > Shading tab

Examples

C#

```
oTableFormatting.Shading.BackgroundColor = Color.Gray;
```

vb.net

```
oTableFormatting.Shading.BackgroundColor = Color.Gray
```

TableFormatting.DefaultSpacing

Description

Returns or sets an `int` that represents the default spacing between a table's cells that has this formatting. This is the distance in twips between the edge of a cell and the center of its border with an adjacent cell. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int DefaultSpacing{ get; set; }
```

Signature: VB.NET

```
Public Property DefaultSpacing() As Integer
```

Remarks

By default, this is set to zero (0) for a new table.

Word equivalent: Table menu > Table Properties... > Table tab > Options... button > Default cell spacing section > Allow spacing between cells

Examples

C#

```
//--- Return DefaultSpacing
          int defaultSpacing = oTableFormatting.DefaultSpacing;

          //--- Set DefaultSpacing to 1/2 inch
          oTableFormatting.DefaultSpacing = 720;
```

vb.net

```
'--- Return DefaultSpacing
          Dim defaultSpacing As Integer = oTableFormatting.DefaultSpacing

          '--- Set DefaultSpacing to 1/2 inch
          oTableFormatting.DefaultSpacing = 720
```

TableFormatting.GetDefaultBorder(Border.Location)

Description

Returns a Border object that represents the default border for table cells at a specified location for a table created with this formatting. Individual table cells can override these border settings. Allowed locations are: Top, Left, Bottom, Right, Vertical, and Horizontal. Top, Left, Bottom, and Right refer to the default border for those sides of a table cell. Vertical and Horizontal refer to the border between table cells.

Signature: C#

```
public Border GetDefaultBorder(Location location)
```

Signature: VB.NET

```
Public Function GetDefaultBorder(ByVal location As Location) As Border
```

Parameters

location

A Border.Location object that represents the border location to get the border at.

Returns

A `Border` object that represents the default border for table cells at a specified location for a table created with this formatting.

Remarks

MS Word equivalent: Format menu > Border and Shading... > Borders tab (With "Apply to: Table" selected)

Examples

C#

```
//--- Return DefaultBorder
          Border oBorder = oTableFormatting.GetDefaultBorder(Border.Location.Top);
```

vb.net

```
'--- Return DefaultBorder
          Dim oBorder As Border = oTableFormatting.GetDefaultBorder(Border.Location.Top)
```

TableFormatting.GetDefaultPadding(TableCell.Location)

Description

Returns an `int` that represents the default table cell margin at a particular location. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public int GetDefaultPadding(Location location)
```

Signature: VB.NET

```
Public Function GetDefaultPadding(ByVal location As Location) As Integer
```

Parameters

location

A TableCell.Location object that represents ths location of the margin.

Returns

An `int` that represents the default table cell margin at a particular location.

Remarks

Allowed locations are Top, Left, Bottom, and Right, which refer to the default margins for those sides of a table cell.

By default, Top and Bottom are set to 0", and Left and Right are set to 0.08".

Word equivalent: Table menu > Table Properties... > Table tab > Options... button > Default cell margins section

Examples

C#

```
//--- Return DefaultPadding for Left margin
          int defaultPadding =
               oTableFormatting.GetDefaultPadding(TableCell.Location.Left);
```

vb.net

```
'--- Return DefaultPadding for Left margin
          Dim defaultPadding As Integer = _
               oTableFormatting.GetDefaultPadding(TableCell.Location.Left)
```

TableFormatting.Justification

Description

Sets or returns a ParagraphFormatting.Justification object that represents the justification for a table created with this formatting. Possible values: Left, Center, Right, LeftRight.

Signature: C#

```
public Justification Justification{ get; set; }
```

Signature: VB.NET

```
Public Property Justification() As Justification
```

Examples

C#

```
//--- Return Justification
          ParagraphFormatting.Justification oJustification =
               oTableFormatting.Justification;

          //--- Set Justification
          oTableFormatting.Justification =
               ParagraphFormatting.Justification.Center;
```

vb.net

```
'--- Return Justification
          Dim oJustification As ParagraphFormatting.Justification = _
               oTableFormatting.Justification

          '--- Set Justification
          oTableFormatting.Justification = _
               ParagraphFormatting.Justification.Center
```

TableFormatting.LeftIndent

Description

Sets or returns an `int` that represents the left indent for a table created with this formatting in twips. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int LeftIndent{ get; set; }
```

Signature: VB.NET

```
Public Property LeftIndent() As Integer
```

Examples

C#

```
//--- Return LeftIndent
          int leftIndent = oTableFormatting.LeftIndent;

          //--- Set LeftIndent to 1 inch
          oTableFormatting.LeftIndent = 1440;
```

vb.net

```
'--- Return LeftIndent
          Dim leftIndent As Integer = oTableFormatting.LeftIndent

          '--- Set LeftIndent to 1 inch
          oTableFormatting.LeftIndent = 1440
```

TableFormatting.RepeatAsHeader

Description

Sets or returns a `boolean` that represents if Word will repeat the first row of a table created with this formatting as a header across multiple pages.

Signature: C#

```
public boolean RepeatAsHeader{ get; set; }
```

Signature: VB.NET

```
Public Property RepeatAsHeader() As Boolean
```

Examples

C#

```
//--- Return RepeatAsHeader
          bool repeatAsHeader = oTableFormatting.RepeatAsHeader;

          //--- Set RepeatAsHeader
          oTableFormatting.RepeatAsHeader = true;
```

vb.net

```
'--- Return RepeatAsHeader
          Dim repeatAsHeader As Boolean = oTableFormatting.RepeatAsHeader

          '--- Set RepeatAsHeader
          oTableFormatting.RepeatAsHeader = True
```

TableFormatting.SetDefaultPadding(TableCell.Location, Int32)

Description

Sets an `int` that represents the default table cell margin at a particular location. One twip = (1/20 pt) or (1/1440 in)

Signature: C#

```
public void SetDefaultPadding(Location location, int value)
```

Signature: VB.NET

```
Public Sub SetDefaultPadding(ByVal location As Location, ByVal value As Integer)
```

Parameters

location

A TableCell.Location object that represents the location at which to set the margin.

value

An `int` that represents the default table cell margin at a particular location in twips.

Remarks

Allowed locations are Top, Left, Bottom, and Right, which refer to the default margins for those sides of a table cell.

By default, Top and Bottom are set to 0", and Left and Right are set to 0.08".

Word equivalent: Table menu > Table Properties... > Table tab > Options... button > Default cell margins section

Examples

C#

```
//--- Set DefaultPadding for Left margin to 1 inch
          oTableFormatting.SetDefaultPadding(TableCell.Location.Left, 1440);
```

vb.net

```
'--- Set DefaultPadding for Left margin to 1 inch
          oTableFormatting.SetDefaultPadding(TableCell.Location.Left, 1440)
```

TwipsConverter

Description

Twips are the units of measurement used by the WordWriter API. A single twip represents 1/1440 of an inch. TwipsConverter is a utility class that converts between twips and other units of measurement.

Signature: C#

```
public static class TwipsConverter
```

Signature: VB.NET

```
Public Shared Class TwipsConverter
```

Examples

C#

```
WordApplication wwApp = new WordApplication();
          Document doc = wwApp.Create();
          Section section = doc.CreateSectionAfter();
          section.PageWidth = TwipsConverter.FromInches(8.5);
          section.PageHeight = TwipsConverter.FromInches(11);
```

vb.net

```
Dim wwApp As New WordApplication()
          Dim doc As Document = wwApp.Create()
          Dim section As Section = doc.CreateSectionAfter()
          section.PageWidth = TwipsConverter.FromInches(8.5)
          section.PageHeight = TwipsConverter.FromInches(11)
```

Methods

Name	Description
FromCentimeters(Double)	Converts centimeters to twips. A twip is 1/567 of a centimeter.
FromInches(Double)	Converts inches to twips. A twip is 1/1440 of an inch.
FromPoints(Double)	Converts points to twips. 1 point = 1/72 inch = 20 twips. 1 inch = 72 points = 1440 twips
ToCentimeters(Int32)	Converts twips to centimeters. A twip is 1/1440 of an inch and 1/567 of a centimeter.
ToInches(Int32)	Converts twips to inches. A twip is 1/1440 of an inch.
ToPoints(Int32)	Converts twips to points. A twip is 1/1440 of an inch and a point is 1/72 of an inch.

TwipsConverter.FromCentimeters(Double)

Description

Converts centimeters to twips. A twip is 1/567 of a centimeter.

Signature: C#

```
public int FromCentimeters(double cm)
```

Signature: VB.NET

```
Public Function FromCentimeters(ByVal cm As Double) As Integer
```

Parameters

cm

The number of centimeters to convert.

Returns

The result of the conversion (twips).

Examples

C#

```
WordApplication wwApp = new WordApplication();
          Document doc = wwApp.Create();
          Section section = doc.CreateSectionAfter();
          section.PageWidth = TwipsConverter.FromCentimeters(21.59);
          section.PageHeight = TwipsConverter.FromCentimeters(27.94);
```

vb.net

```
Dim wwApp As New WordApplication()
          Dim doc As Document = wwApp.Create()
          Dim section As Section= doc.CreateSectionAfter()
          section.PageWidth = TwipsConverter.FromCentimeters(21.59)
          section.PageHeight = TwipsConverter.FromCentimeters(27.94)
```

TwipsConverter.FromInches(Double)

Description

Converts inches to twips. A twip is 1/1440 of an inch.

Signature: C#

```
public int FromInches(double inches)
```

Signature: VB.NET

```
Public Function FromInches(ByVal inches As Double) As Integer
```

Parameters

inches

The number of inches to convert.

Returns

The result of the conversion (twips).

Examples

C#

```
WordApplication wwApp = new WordApplication();
          Document doc = wwApp.Create();
          Section section = doc.CreateSectionAfter();
          section.PageWidth = TwipsConverter.FromInches(8.5);
          section.PageHeight = TwipsConverter.FromInches(11);
```

vb.net

```
Dim wwApp As New WordApplication()
          Dim doc As Document = wwApp.Create()
          Dim section As Section = doc.CreateSectionAfter()
          section.PageWidth = TwipsConverter.FromInches(8.5)
          section.PageHeight = TwipsConverter.FromInches(11)
```

TwipsConverter.FromPoints(Double)

Description

Converts points to twips. 1 point = 1/72 inch = 20 twips.
1 inch = 72 points = 1440 twips

Signature: C#

```
public int FromPoints(double pts)
```

Signature: VB.NET

```
Public Function FromPoints(ByVal pts As Double) As Integer
```

Parameters

pts

The number of points to convert.

Returns

The result of the conversion (twips).

Examples

C#

```
WordApplication wwApp = new WordApplication();
          Document doc = wwApp.Create();
          Section section = doc.CreateSectionAfter();
          section.PageWidth = TwipsConverter.FromPoints(612);
          section.PageHeight = TwipsConverter.FromPoints(792);
```

vb.net

```
Dim wwApp As New WordApplication()
          Dim doc As Document = wwApp.Create()
          Dim section As Section = doc.CreateSectionAfter()
          section.PageWidth = TwipsConverter.FromPoints(612)
          section.PageHeight = TwipsConverter.FromPoints(792)
```

TwipsConverter.ToCentimeters(Int32)

Description

Converts twips to centimeters. A twip is 1/1440 of an inch and 1/567 of a centimeter.

Signature: C#

```
public double ToCentimeters(int twips)
```

Signature: VB.NET

```
Public Function ToCentimeters(ByVal twips As Integer) As Double
```

Parameters

twips

The number of twips to convert to centimeters.

Returns

The result of the conversion (centimeters).

Examples

C#

```
TwipsConverter.ToCentimeters(12240);
```

vb.net

```
TwipsConverter.ToCentimeters(12240)
```

TwipsConverter.ToInches(Int32)

Description

Converts twips to inches. A twip is 1/1440 of an inch.

Signature: C#

```
public double ToInches(int twips)
```

Signature: VB.NET

```
Public Function ToInches(ByVal twips As Integer) As Double
```

Parameters

twips

The number of twips to convert to inches.

Returns

The result of the conversion (inches).

Examples

C#
`TwipsConverter.ToInches(12240);`

vb.net
`TwipsConverter.ToInches(12240)`

TwipsConverter.ToPoints(Int32)

Description

Converts twips to points. A twip is 1/1440 of an inch and a point is 1/72 of an inch.

Signature: C#

```
public double ToPoints(int twips)
```

Signature: VB.NET

```
Public Function ToPoints(ByVal twips As Integer) As Double
```

Parameters

twips

The number of twips to convert to centimeters.

Returns

The result of the conversion (points).

Examples

C#
`TwipsConverter.ToPoints(12240);`

vb.net
`TwipsConverter.ToPoints(12240)`

Units

Introduced in build 4.1.0.1331

Description

The different types of units used by WordWriter

Signature: C#

```
public enum Units
```

Signature: VB.NET

```
public Enum Units
```

Remarks

Not all units are supported by all methods/properties, check the documentation before selecting units.

Values

Name	Description
Null	Default unit; associated size is ignored
Auto	Size is automatically chosen; associated size is ignored
Percent	Size is measured as a percent of the total available width
Twips	Size is measured in twips

Watermark

Description

A Watermark object represents a document's watermark, which is an image or text that when printed appears in the background of a document. The property Document.Watermark returns a `Watermark` object.

Signature: C#

```
public sealed class Watermark
```

Signature: VB.NET

```
Public NotInheritable Class Watermark
```

Examples

C#

```
WordApplication app = new WordApplication();
Document doc = app.Create();
Watermark wmark = doc.Watermark;
```

vb.net

```
Dim app As New WordApplication()
Dim doc As Document = app.Create()
Dim wmark As Watermark = doc.Watermark
```

Properties

Name	Description
Color	Sets or returns the color of the watermark.
FontName	Sets or returns the font of watermark text.
Image	Sets or returns an image to use as the watermark.
Scale	Sets or returns the size of a watermark image as a percentage of its actual size in the source file.
Semitransparent	Sets or returns whether a text watermark will be semitransparent or not.
Size	Sets or returns the size of watermark text.
Text	Sets or returns text to use as the document's watermark.
Washout	Sets or returns whether an image watermark will be faded or not.
WatermarkLayout	Sets or returns the layout of watermark text.
WatermarkType	Returns the watermark's type, which may be image, text, or none.

Methods

Name	Description
ClearWatermark()	Clears all properties assigned to the watermark. Call this method to remove a watermark from the document.

Nested Classes

Name	Description
Layout	Used to specify the layout for a text watermark.
Type	Used to specify a type of watermark.

Watermark.ClearWatermark()

Description

Clears all properties assigned to the watermark.
Call this method to remove a watermark from the document.

Signature: C#

```
public void ClearWatermark()
```

Signature: VB.NET

```
Public Sub ClearWatermark()
```

Watermark.Color

Description

Sets or returns the color of the watermark.

Signature: C#

```
public System.Drawing.Color Color{ get; set; }
```

Signature: VB.NET

```
Public Property Color() As System.Drawing.Color
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;

          //--- Get Color
          System.Drawing.Color wmarkColor = wmark.Color;

          //--- Set Color
          wmark.Color = System.Drawing.Color.Blue;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark

          '--- Get Color
          Dim wmarkColor As System.Drawing.Color = wmark.Color

          '--- Set Color
          wmark.Color = System.Drawing.Color.Blue
```

Watermark.FontName

Description

Sets or returns the font of watermark text.

Signature: C#

```
public System.String FontName{ get; set; }
```

Signature: VB.NET

```
Public Property FontName() As String
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          wmark.Text = "CONFIDENTIAL";
          wmark.FontName = "Arial";
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          wmark.Text = "CONFIDENTIAL"
          wmark.FontName = "Arial"
```

Watermark.Image

Description

Sets or returns an image to use as the watermark.

Signature: C#

```
public System.IO.Stream Image{ get; set; }
```

Signature: VB.NET

```
Public Property Image() As System.IO.Stream
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          wmark.Image = File.Open("somepic.gif");
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          wmark.Image = File.Open("somepic.gif")
```

Watermark.Layout

Description

Used to specify the layout for a text watermark.

Signature: C#

```
public sealed class Layout : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Layout
  Inherits TypesafeEnum
```

Examples

C#

```
WordApplication app = new WordApplication();
            Document doc = app.Create();
            Watermark wmark = doc.Watermark;
            wmark.Text = "CONFIDENTIAL";
            wmark.WatermarkLayout = Watermark.Layout.Diagonal;
```

vb.net

```
Dim app As New WordApplication()
            Dim doc As Document = app.Create()
            Dim wmark As Watermark = doc.Watermark
            wmark.Text = "CONFIDENTIAL"
            wmark.WatermarkLayout = Watermark.Layout.Diagonal
```

Values

Name	Description
Diagonal	Text is diagonal
Horizontal	Text is horizontal

Watermark.Scale

Description

Sets or returns the size of a watermark image as a percentage of its actual size in the source file.

Signature: C#

```
public int Scale{ get; set; }
```

Signature: VB.NET

```
Public Property Scale() As Integer
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          wmark.Image = File.Open("somepic.gif");
          wmark.Scale = 75;
```

C#

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          wmark.Image = File.Open("somepic.gif")
          wmark.Scale = 75
```

Watermark.Semitransparent

Description

Sets or returns whether a text watermark will be semitransparent or not.

Signature: C#

```
public boolean Semitransparent{ get; set; }
```

Signature: VB.NET

```
Public Property Semitransparent() As Boolean
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          wmark.Text = "CONFIDENTIAL";
          wmark.Semitransparent = true;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          wmark.Text = "CONFIDENTIAL"
          wmark.Semitransparent = True
```

Watermark.Size

Description

Sets or returns the size of watermark text.

Signature: C#

```
public double Size{ get; set; }
```

Signature: VB.NET

```
Public Property Size() As Double
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          wmark.Text = "CONFIDENTIAL";
          wmark.Size = 36;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          wmark.Text = "CONFIDENTIAL"
          wmark.Size = 36
```

Watermark.Text

Description

Sets or returns text to use as the document's watermark.

Signature: C#

```
public System.String Text{ get; set; }
```

Signature: VB.NET

```
Public Property Text() As String
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          wmark.Text = "CONFIDENTIAL";
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          wmark.Text = "CONFIDENTIAL"
```

Watermark.Type

Description

Used to specify a type of watermark.

Signature: C#

```
public sealed class Type : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Type
  Inherits TypesafeEnum
```

Examples

C#

```
WordApplication app = new WordApplication();
            Document doc = app.Create();
            Watermark wmark = doc.Watermark;
            Watermark.Type markType = wmark.WatermarkType;
```

vb.net

```
Dim app As New WordApplication()
            Dim doc As Document = app.Create()
            Watermark wmark = doc.Watermark
            Watermark.Type markType = wmark.WatermarkType
```

Values

Name	Description
Image	Image Watermark
None	No Watermark
Thick	Text Watermark

Watermark.Washout

Description

Sets or returns whether an image watermark will be faded or not.

Signature: C#

```
public boolean Washout{ get; set; }
```

Signature: VB.NET

```
Public Property Washout() As Boolean
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          wmark.Image = File.Open("somepic.gif");
          wmark.Washout = true;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          wmark.Image = File.Open("somepic.gif")
          wmark.Washout = True
```

Watermark.WatermarkLayout

Description

Sets or returns the layout of watermark text.

Signature: C#

```
public Layout WatermarkLayout{ get; set; }
```

Signature: VB.NET

```
Public Property WatermarkLayout() As Layout
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          wmark.Text = "CONFIDENTIAL";
          wmark.WatermarkLayout = Watermark.Layout.Diagonal;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          wmark.Text = "CONFIDENTIAL"
          wmark.WatermarkLayout = Watermark.Layout.Diagonal
```

Watermark.WatermarkType

Description

Returns the watermark's type, which may be image, text, or none.

Signature: C#

```
public Type WatermarkType{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property WatermarkType() As Type
```

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Create();
          Watermark wmark = doc.Watermark;
          Watermark.Type markType = wmark.WatermarkType;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Create()
          Dim wmark As Watermark = doc.Watermark
          Dim markType As Watermark.Type = wmark.WatermarkType
```

WordApplication

Description

WordApplication is the main class for pure code-based Word document generation (for information about template-based workbook generation, see WordTemplate). This class is an engine used to open, create, and write (save to disk or stream to a browser) Word documents. A single instance of `WordApplication` can generate multiple documents.

Signature: C#

```
public sealed class WordApplication
```

Signature: VB.NET

```
Public NotInheritable Class WordApplication
```

Remarks

A generated document is represented by the Document class.

The `WordApplication` class is in the `SoftArtisans.OfficeWriter.WordWriter` namespace. The class can can be referenced as `SoftArtisans.OfficeWriter.WordWriter`. To minimize typing and errors, import the namespace to the aspx page, and reference the class as `WordApplication`, without the namespace prefix. If you are coding directly in the .aspx page, following the `Page` directive, include:

If you are coding in the code-behind page (.aspx.cs or .aspx.vb), include a `using` or `Imports` statement at the top of the code behind page:

C#

```
using SoftArtisans.OfficeWriter.WordWriter;
```

vb.net

```
Imports SoftArtisans.OfficeWriter.WordWriter
```

Examples

C#

```
//--- Create an instance of the WordApplication class
          WordApplication oWordApplication = new WordApplication();
```

vb.net

```
'--- Create an instance of the WordApplication class
          Dim oWordApplication As New WordApplication()
```

Properties

Name	Description
AlwaysUseUnicode	Returns a boolean that represents whether or not WordWriter will write all text using unicode. A value of `false` can result in smaller file sizes but text that is not part of cp1252 (Windows-1252) will be unreadable. This is set to `true` by default. It is recommended that you should set this to `false` if it is known that only characters supported by Windows-1252 are in the document, especially when creating large documents.
LicenseKey	Sets or returns the license key used for this instance of WordApplication. This property is mainly used for debugging purposes. For managing your license keys, please use the License Key Manager that is installed with OfficeWriter.
Preserve	Sets or returns a boolean that represents whether or not WordWriter will preserve feature of an existing document that it does not natively support. Setting this to false can result in smaller file sizes, but features that are not natively supported by WordWriter will be stripped out. This is set to true by default.
Version	Returns the exact version of WordWriter. The format of the version string returned is:

Methods

Name	Description
Create()	Creates and returns a new Document based on the Normal style.

Open(String)	Opens and returns an existing Document from disk.
Open(System.IO.Stream)	Opens and returns an existing Document from the specified `Stream`.
Open(WordTemplate)	Takes a WordTemplate object representing a template Word file, and opens the file as a Document object.
Save(Document, String)	Writes a Document to a file.
Save(Document, System.IO.Stream)	Writes a Document to a stream in the form of a Microsoft Word document.
Save(Document, System.Web.HttpResponse, String, Boolean)	Streams a Document to an HttpResponse with the option of having it open in the user's web browser. The fileName specified will appear in the browser window. In addition, the content-type to use when streaming the file is specified.
Save(Document, System.Web.HttpResponse, String, Boolean, String)	Streams a Document to an HttpResponse with the option of having it open in the user's web browser. The fileName specified will appear in the browser window. In addition, the content-type to use when streaming the file is specified.
WordApplication()	Creates a new WordApplication object

Extension Methods

Overload	Description
Open(Microsoft.SharePoint.SPDocumentLibrary, String)	Opens and returns an existing Document from a SharePoint Document Library.
Open(Microsoft.SharePoint.SPListItem, String)	Opens and returns an existing Document from a SharePoint List Item.
Save(Document, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Writes a Document to a file and saves it to a SharePoint Document Library.
Save(Document, Microsoft.SharePoint.SPListItem, String)	Writes a Document to a file and saves it to a SharePoint List Item.

WordApplication.AlwaysUseUnicode

Description

Returns a boolean that represents whether or not WordWriter will write all text using unicode. A value of `false` can result in smaller file sizes but text that is not part of cp1252 (Windows-1252) will be unreadable. This is set to `true` by default. It is recommended that you should set this to `false` if it is known that only characters supported by Windows-1252 are in the document, especially when creating large documents.

Signature: C#

```
public boolean AlwaysUseUnicode{ get; set; }
```

Signature: VB.NET

```
Public Property AlwaysUseUnicode() As Boolean
```

Examples

C#

```
//--- Return AlwaysUseUnicode
          bool alwaysUseUnicode = oWordApplication.AlwaysUseUnicode;

          //--- Set AlwaysUseUnicode
          oWordApplication.AlwaysUseUnicode = false;
```

vb.net

```
'--- Return AlwaysUseUnicode
          Dim alwaysUseUnicode As Boolean = oWordApplication.AlwaysUseUnicode

          '--- Set AlwaysUseUnicode
          oWordApplication.AlwaysUseUnicode = False
```

WordApplication.Create()

Description

Creates and returns a new Document based on the Normal style.

Signature: C#

```
public Document Create()
```

Signature: VB.NET

```
Public Function Create() As Document
```

Returns

A new document based on the Normal style.

Exceptions

System.Exception

Thrown if an error occurs creating the document.

Remarks

MS Word equivalent: File > New... > Blank Document

Examples

C#

```
Document doc = app.Create();
```

vb.net

```
Dim doc As Document = app.Create()
```

WordApplication.LicenseKey

Description

Sets or returns the license key used for this instance of WordApplication. This property is mainly used for debugging purposes. For managing your license keys, please use the License Key Manager that is installed with OfficeWriter.

Signature: C#

```
public System.String LicenseKey{ get; set; }
```

Signature: VB.NET

```
Public Property LicenseKey() As String
```

Examples

C#

```
//--- Return license key
          string key = oWordApplication.LicenseKey;

          //--- Set license key
          oWordApplication.LicenseKey = "XXXXXX-XXXX-XXXX-XXXXXX";
```

vb.net

```
'--- Return license key
          Dim key As String = oWordApplication.LicenseKey

          '--- Set license key
          oWordApplication.LicenseKey = "XXXXXX-XXXX-XXXX-XXXXXX"
```

WordApplication.Open

Overload	Description
Open(System.IO.Stream)	Opens and returns an existing Document from the specified `Stream`.
Open(String)	Opens and returns an existing Document from disk.
Open(WordTemplate)	Takes a WordTemplate object representing a template Word file, and opens the file as a Document object.

SharePoint Integration Extension Methods

Overload	Description
Open(Microsoft.SharePoint.SPDocumentLibrary, String)	Opens and returns an existing Document from a SharePoint Document Library.
Open(Microsoft.SharePoint.SPListItem, String)	Opens and returns an existing Document from a SharePoint List Item.

WordApplication.Open(String)

Description

Opens and returns an existing Document from disk.

Signature: C#

```
public Document Open(System.String fileName)
```

Signature: VB.NET

```
Public Function Open(ByVal fileName As String) As Document
```

Parameters

fileName

The complete path and filename of the document to open from disk.

Returns

The document that is opened from disk.

Exceptions

System.Exception

Thrown if an error occurs while opening the document.

Remarks

Though the WordTemplate object currently supports Office Open XML (Word 2007) format files, WordApplication.Open() does not yet support them.

Examples

C#

```
Document doc = app.Open("@C:\sample.doc");
```

vb.net

```
Dim doc As Document = app.Open("C:\sample.doc")
```

WordApplication.Open(System.IO.Stream)

Description

Opens and returns an existing Document from the specified `Stream`.

Signature: C#

```
public Document Open(System.IO.Stream stream)
```

Signature: VB.NET

```
Public Function Open(ByVal stream As System.IO.Stream) As Document
```

Parameters

stream

Stream to read the document from.

Returns

The document contained in the specified stream.

Exceptions

System.Exception

Thrown if there is an error opening the document.

Remarks

Though the WordTemplate object currently supports Office Open XML (Word 2007) format files, WordApplication.Open() does not yet support them.

Examples

C#

```
Document doc = app.Open(strm);
```

vb.net

```
Dim doc As Document = app.Open(strm)
```

WordApplication.Open(WordTemplate)

Description

Takes a WordTemplate object representing a template Word file, and opens the file as a Document object.

Signature: C#

```
public Document Open(WordTemplate wwt)
```

Signature: VB.NET

```
Public Function Open(ByVal wwt As WordTemplate) As Document
```

Parameters

wwt

A `WordTemplate` object representing a template Word file.

Returns

The Word file that was passed from `WordTemplate` to WordApplication.

Exceptions

System.Exception

Thrown if an error occurs while opening the template.

Remarks

Though the `WordTemplate` object currently supports Office Open XML (Word 2007) format files, WordApplication.Open() does not yet support them.

Examples

C#

```
WordTemplate wTempl = new WordTemplate();
          ...
          wTempl.Process();
          WordApplication wApp = new WordApplication();
          Document doc = wApp.Open(wTempl);
          ...
```

vb.net

```
Dim wTempl As New WordTemplate()
          ...
          wTempl.Process()
          Dim wApp As New WordApplication()
          Dim doc As Document = wApp.Open(wTempl)
```

WordApplication.Preserve

Description

Sets or returns a boolean that represents whether or not WordWriter will preserve feature of an existing document that it does not natively support. Setting this to false can result in smaller file sizes, but features that are not natively supported by WordWriter will be stripped out. This is set to true by default.

Signature: C#

```
public boolean Preserve{ get; set; }
```

Signature: VB.NET

```
Public Property Preserve() As Boolean
```

Remarks

The main advantage to setting the `preserve` property to false is that it can significantly reduce the file size of the resulting Word document. This is recommended if it is known that only supported features are in the document, especially when creating large documents.

Examples

C#

```
//--- Return Preserve
          bool preserve = oWordApplication.Preserve;

          //--- Set Preserve
          oWordApplication.Preserve = false;
```

vb.net

```
'--- Return Preserve
          Dim preserve As Boolean = oWordApplication.Preserve

          '--- Set Preserve
          oWordApplication.Preserve = False
```

WordApplication.Save

Overload	Description
Save(Document, String)	Writes a Document to a file.
Save(Document, System.IO.Stream)	Writes a Document to a stream in the form of a Microsoft Word document.
Save(Document, System.Web.HttpResponse, String, Boolean)	Streams a Document to an HttpResponse with the option of having it open in the user's web browser. The fileName specified will appear in the browser window. In addition, the content-type to use when streaming the file is specified.
Save(Document, System.Web.HttpResponse, String, Boolean, String)	Streams a Document to an HttpResponse with the option of having it open in the user's web browser. The fileName specified will appear in the browser window. In addition, the content-type to use when streaming the file is specified.

SharePoint Integration Extension Methods

Overload	Description
Save(Document, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Writes a Document to a file and saves it to a SharePoint Document Library.
Save(Document, Microsoft.SharePoint.SPListItem, String)	Writes a Document to a file and saves it to a SharePoint List Item.

WordApplication.Save(Document, String)

Description

Writes a Document to a file.

Signature: C#

```
public void Save(Document doc, System.String fileName)
```

Signature: VB.NET

```
Public Sub Save(ByVal doc As Document, ByVal fileName As String)
```

Parameters

doc

The `Document` to save.

fileName

The full filename to save the `Document` to.

Exceptions

System.Exception

Thrown if there is an error saving the file.

Remarks

The WordApplication.Preserve property is used to specify the way WordWriter will save the document. When the `preserve` property is set to true, WordWriter attempts to preserve all formatting and features that it doesn't directly support. This includes Fields, Footnotes, Hyperlinks, Comments, and Anchored Images. When the `preserve` property is false, only those features directly supported by WordWriter will be preserved. This includes all Tables, Lists, Headers and Footers.

Examples

C#

```
app.Preserve = true;
          app.Save(doc, "@C:\sample.doc");
```

vb.net

```
app.Preserve = True
          app.Save(doc, "C:\sample.doc")
```

WordApplication.Save(Document, System.IO.Stream)

Description

Writes a Document to a stream in the form of a Microsoft Word document.

Signature: C#

```
public void Save(Document doc, System.IO.Stream @out)
```

Signature: VB.NET

```
Public Sub Save(ByVal doc As Document, ByVal out As System.IO.Stream)
```

Parameters

doc

The `Document` to save.

out

The `Stream` to write the file to.

Exceptions

System.Exception

Thrown if there is a problem saving the file.

Remarks

The WordApplication.Preserve property is used to specify the way WordWriter will save the document. When the `preserve` property is set to true, WordWriter attempts to preserve all formatting and features that it doesn't directly support. This includes Fields, Footnotes, Hyperlinks, Comments, and Anchored Images. When the `preserve` property is false, only those features directly supported by WordWriter will be preserved. This includes all Tables, Lists, Headers and Footers.

Examples

C#

```
app.Save(doc, strm);
```

vb.net

```
app.Save(doc, strm)
```

WordApplication.Save(Document, System.Web.HttpResponse, String, Boolean, String)

Description

Streams a Document to an HttpResponse with the option of having it open in the user's web browser. The fileName specified will appear in the browser window. In addition, the content-type to use when streaming the file is specified.

Signature: C#

```
public void Save(Document doc, System.Web.HttpResponse response, System.String fileName, boolean
openInBrowser, System.String contentType)
```

Signature: VB.NET

```
Public Sub Save(ByVal doc As Document, ByVal response As System.Web.HttpResponse, ByVal fileName As
String, ByVal openInBrowser As Boolean, ByVal contentType As String)
```

Parameters

doc

The `Document` to save.

response

The HTTPResponse to write the document to.

fileName

The filename to display in the browser window.

openInBrowser

Whether to open the document in the browser window (true) or have it open in Word (false).

contentType

The content-type to stream the file as.

Exceptions

System.Exception

Thrown if there is an error streaming the document.

Remarks

For the document to open in the browser window, the "Browse in same window" option must be enabled for .doc file types on the client machine.

The WordApplication.Preserve property is used to specify the way WordWriter will save the document. When the `preserve` property is set to true, WordWriter attempts to preserve all formatting and features that it doesn't directly support. This includes Fields, Footnotes, Hyperlinks, Comments, and Anchored Images. When the `preserve` property is false, only those features directly supported by WordWriter will be preserved. This includes all Tables, Lists, Headers and Footers.

Examples

C#

```
app.Preserve = true;
          app.Save(doc, Page.Response, "sample.doc", false, "application/x-msdownload");
```

vb.net

```
app.Preserve = True
          app.Save(doc, Page.Response, "sample.doc", False, "application/x-msdownload")
```

WordApplication.Save(Document, System.Web.HttpResponse, String, Boolean)

Description

Streams a Document to an HttpResponse with the option of having it open in the user's web browser. The fileName specified will appear in the browser window. In addition, the content-type to use when streaming the file is specified.

Signature: C#

```
public void Save(Document doc, System.Web.HttpResponse response, System.String fileName, boolean
openInBrowser)
```

Signature: VB.NET

```
Public Sub Save(ByVal doc As Document, ByVal response As System.Web.HttpResponse, ByVal fileName As
String, ByVal openInBrowser As Boolean)
```

Parameters

doc

The `Document` to save.

response

The HTTPResponse to write the document to.

fileName

The filename to display in the browser window.

openInBrowser

Whether to open the document in the browser window (true) or have it open in Word (false).

Exceptions

System.Exception

Thrown if there is an error streaming the document.

Remarks

For the document to open in the browser window, the "Browse in same window" option must be enabled for .doc file types on the client machine.

The WordApplication.Preserve property is used to specify the way WordWriter will save the document. When the `preserve` property is set to true, WordWriter attempts to preserve all formatting and features that it doesn't directly support. This includes Fields, Footnotes, Hyperlinks, Comments, and Anchored Images. When the `preserve` property is false, only those features directly supported by WordWriter will be preserved. This includes all Tables, Lists, Headers and Footers.

Examples

C#

```
oWordApplication.Preserve = true;
          oWordApplication.Save(oDocument, Page.Response, "sample.doc", true);
```

vb.net

```
oWordApplication.Preserve = True
          oWordApplication.Save(oDocument, Page.Response, "sample.doc", True)
```

WordApplication.Version

Description

Returns the exact version of WordWriter. The format of the version string returned is:

Signature: C#

```
public System.String Version{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Version() As String
```

Remarks

Version is a **Read Only** property.

Examples

C#

```
string ver = oWordApplication.Version;
```

vb.net

```
Dim ver As String = oWordApplication.Version
```

WordApplication.WordApplication()

Description

Creates a new WordApplication object

Signature: C#

```
public WordApplication ( )
```

Signature: VB.NET

```
Public Sub New WordApplication ( )
```

Examples

C#

```
WordApplication wa = new WordApplication();
```

vb.net

```
Dim wa As New WordApplication()
```

WordChangeTracking

Description

This class is used to get and set Word change tracking settings for a document.

Signature: C#

```
public sealed class WordChangeTracking
```

Signature: VB.NET

```
Public NotInheritable Class WordChangeTracking
```

Remarks

You cannot create a new WordChangeTracking object. You must use Document.ChangeTracking to retrieve WordChangeTracking properties for a document.

The following sample gets the `WordChangeTracking` object of a document so you can start working with its properties.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          WordChangeTracking changeTracking = doc.ChangeTracking;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim changeTracking As WordChangeTracking = doc.ChangeTracking
```

Properties

Name	Description
DisplayRevisions	Sets or returns a `boolean` that represents whether or not Word will display revisions for this document.
LockRevisionMarkingState	Sets or returns a `boolean` that represents whether or not the current revision tracking state is locked.
PrintRevisions	Sets or returns a `boolean` that represents whether or not Word will print revision tracking markup when this document is printed.

ShowAnnotationRevisions	Sets or returns a `boolean` that represents whether or not Word will show annotations in this document.
ShowFormattingRevisions	Sets or returns a `boolean` that represents whether or not Word will show formatting revisions in this document.
ShowInkRevisions	Sets or returns a `boolean` that represents whether or not Word will show ink revisions in this document.
ShowInsertDeleteRevisions	Sets or returns a `boolean` that represents whether or not Word will show insert and delete revisions in this document.
ShowMarkupRevisions	Sets or returns a `boolean` that represents whether or not Word will show revision markup in this document.
TrackChanges	Sets or returns a `boolean` that represents whether or not Word will track changes for this document.

WordChangeTracking.DisplayRevisions

Description

Sets or returns a `boolean` that represents whether or not Word will display revisions for this document.

Signature: C#

```
public boolean DisplayRevisions{ get; set; }
```

Signature: VB.NET

```
Public Property DisplayRevisions() As Boolean
```

Remarks

This is set to true by default if TrackChanges is set to true.

Examples

C#

```
//--- Return DisplayRevisions
          bool displayRevisions = changeTracking.DisplayRevisions;

          //--- Set DisplayRevisions
          changeTracking.DisplayRevisions = false;
```

vb.net

```
'--- Return DisplayRevisions
          Dim displayRevisions As Boolean = changeTracking.DisplayRevisions

          '--- Set DisplayRevision
          changeTracking.DisplayRevisions = False
```

WordChangeTracking.LockRevisionMarkingState

Description

Sets or returns a `boolean` that represents whether or not the current revision tracking state is locked.

Signature: C#

```
public boolean LockRevisionMarkingState{ get; set; }
```

Signature: VB.NET

```
Public Property LockRevisionMarkingState() As Boolean
```

Examples

C#

```
//--- Return LockRevisionMarkingState
          bool lockRevisionMarkingState = changeTracking.LockRevisionMarkingState;

          //--- Set LockRevisionMarkingState
          changeTracking.LockRevisionMarkingState = true;
```

vb.net

```
'--- Return LockRevisionMarkingState
          Dim lockRevisionMarkingState As Boolean = changeTracking.LockRevisionMarkingState

          '--- Set LockRevisionMarkingState
          changeTracking.LockRevisionMarkingState = True
```

WordChangeTracking.PrintRevisions

Description

Sets or returns a `boolean` that represents whether or not Word will print revision tracking markup when this document is printed.

Signature: C#

```
public boolean PrintRevisions{ get; set; }
```

Signature: VB.NET

```
Public Property PrintRevisions() As Boolean
```

Examples

C#

```
//--- Return PrintRevisions
          bool printRevisions = changeTracking.PrintRevisions;

          //--- Set PrintRevisions
          changeTracking.PrintRevisions = true;
```

vb.net

```
'--- Return PrintRevisions
          Dim printRevisions As Boolean = changeTracking.PrintRevisions

          '--- Set PrintRevisions
          changeTracking.PrintRevisions = True
```

WordChangeTracking.ShowAnnotationRevisions

Description

Sets or returns a `boolean` that represents whether or not Word will show annotations in this document.

Signature: C#

```
public boolean ShowAnnotationRevisions{ get; set; }
```

Signature: VB.NET

```
Public Property ShowAnnotationRevisions() As Boolean
```

Remarks

This option is set to true by default.

Examples

C#

```
//--- Return ShowAnnotationRevisions
          bool showAnnotationRevisions = changeTracking.ShowAnnotationRevisions;

          //--- Set ShowAnnotationRevisions
          changeTracking.ShowAnnotationRevisions = false;
```

vb.net

```
'--- Return ShowAnnotationRevisions
          Dim showAnnotationRevisions As Boolean = changeTracking.ShowAnnotationRevisions

          '--- Set ShowAnnotationRevisions
          changeTracking.ShowAnnotationRevisions = False
```

WordChangeTracking.ShowFormattingRevisions

Description

Sets or returns a `boolean` that represents whether or not Word will show formatting revisions in this document.

Signature: C#

```
public boolean ShowFormattingRevisions{ get; set; }
```

Signature: VB.NET

```
Public Property ShowFormattingRevisions() As Boolean
```

Remarks

This option is set to false by default.

MS Word equivalent: Tools menu > Options... > Track changes tab > Markup section > Formatting:

Examples

C#

```
//--- Return ShowFormattingRevisions
          bool showFormattingRevisions = changeTracking.ShowFormattingRevisions;

          //--- Set ShowFormattingRevisions
          changeTracking.ShowFormattingRevisions = true;
```

vb.net

```
'--- Return ShowFormattingRevisions
          Dim showFormattingRevisions As Boolean = changeTracking.ShowFormattingRevisions

          '--- Set ShowFormattingRevisions
          changeTracking.ShowFormattingRevisions = True
```

WordChangeTracking.ShowInkRevisions

Description

Sets or returns a `boolean` that represents whether or not Word will show ink revisions in this document.

Signature: C#

```
public boolean ShowInkRevisions{ get; set; }
```

Signature: VB.NET

```
Public Property ShowInkRevisions() As Boolean
```

Examples

C#

```
//--- Return ShowInkRevisions
          bool showInkRevisions = changeTracking.ShowInkRevisions;

          //--- Set ShowInkRevisions
          changeTracking.ShowInkRevisions = true;
```

vb.net

```
'--- Return ShowInkRevisions
          Dim showInkRevisions As Boolean = changeTracking.ShowInkRevisions

          '--- Set ShowInkRevisions
          changeTracking.ShowInkRevisions = True
```

WordChangeTracking.ShowInsertDeleteRevisions

Description

Sets or returns a `boolean` that represents whether or not Word will show insert and delete revisions in this document.

Signature: C#

```
public boolean ShowInsertDeleteRevisions{ get; set; }
```

Signature: VB.NET

```
Public Property ShowInsertDeleteRevisions() As Boolean
```

Remarks

This option is set to true by default.

MS Word equivalent: Tools menu > Options... > Track changes tab > Markup section > Insertions: and Deletions:

Examples

C#

```
//--- Return ShowInsertDeleteRevisions
          bool showInsertDeleteRevisions = changeTracking.ShowInsertDeleteRevisions;

          //--- Set ShowInsertDeleteRevisions
          changeTracking.ShowInsertDeleteRevisions = false;
```

vb.net

```
'--- Return ShowInsertDeleteRevisions
          Dim showInsertDeleteRevisions As Boolean = changeTracking.ShowInsertDeleteRevisions

          '--- Set ShowInsertDeleteRevisions
          changeTracking.ShowInsertDeleteRevisions = False
```

WordChangeTracking.ShowMarkupRevisions

Description

Sets or returns a `boolean` that represents whether or not Word will show revision markup in this document.

Signature: C#

```
public boolean ShowMarkupRevisions{ get; set; }
```

Signature: VB.NET

```
Public Property ShowMarkupRevisions() As Boolean
```

Remarks

This option is set to true by default.

Examples

C#

```
//--- Return ShowMarkupRevisions
          bool showMarkupRevisions = changeTracking.ShowMarkupRevisions;

          //--- Set ShowMarkupRevisions
          changeTracking.ShowMarkupRevisions = false;
```

vb.net

```
'--- Return ShowMarkupRevisions
          Dim showMarkupRevisions As Boolean = changeTracking.ShowMarkupRevisions

          '--- Set ShowMarkupRevisions
          changeTracking.ShowMarkupRevisions = False
```

WordChangeTracking.TrackChanges

Description

Sets or returns a `boolean` that represents whether or not Word will track changes for this document.

Signature: C#

```
public boolean TrackChanges{ get; set; }
```

Signature: VB.NET

```
Public Property TrackChanges() As Boolean
```

Remarks

This is set to false by default.

MS Word equivalent: Tools menu > Track Changes

Examples

C#

```
//--- Return TrackChanges
          bool trackChanges = changeTracking.TrackChanges;

          //--- Set TrackChanges
          changeTracking.TrackChanges = true;
```

vb.net

```
'--- Return TrackChanges
          Dim trackChanges As Boolean = changeTracking.TrackChanges

          '--- Set TrackChanges
          changeTracking.TrackChanges = True
```

WordHyphenSettings

Description

This class is used to get and set Word hyphen settings for a document.

Signature: C#

```
public sealed class WordHyphenSettings
```

Signature: VB.NET

```
Public NotInheritable Class WordHyphenSettings
```

Remarks

You cannot create a new `WordHyphenSettings` object. You must use the Document.HyphenSettings property to retrieve WordHyphenSettings properties for a document.

The following sample gets the `WordHyphenSettings` object of a document so you can start working with these properties.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          WordHyphenSettings hyphenSettings = doc.HyphenSettings;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim hyphenSettings As WordHyphenSettings = doc.HyphenSettings
```

Properties

Name	Description
AllowedConsecutiveHyphens	Sets or returns an `int` that represents the number of consecutive lines that can be auto-hyphenated by Word in this document.
AutoHyphen	Sets or returns a `boolean` that represents whether or not Word will automatically hyphenate this document.
CapitalHyphenation	Sets or returns a `boolean` that represents whether or not Word is allowed to hyphenate words that are capitalized in this document.
HyphenationWidth	Sets or returns an `int` that represents the hyphenation width in twips for this document. The hyphenation width is the amount of space to leave between the end of the last word in a line and the right margin. To reduce the number of hyphens, make the hyphenation zone wider. To reduce the raggedness of the right margin, make the hyphenation zone narrower. One twip = (1/20 pt) or (1/1440 in).

WordHyphenSettings.AllowedConsecutiveHyphens

Description

Sets or returns an `int` that represents the number of consecutive lines that can be auto-hyphenated by Word in this document.

Signature: C#

```
public int AllowedConsecutiveHyphens{ get; set; }
```

Signature: VB.NET

```
Public Property AllowedConsecutiveHyphens() As Integer
```

Remarks

This option is set to No limit by default.

MS Word equivalent: Tools menu > Language > Hyphenation... > Limit consecutive hyphens to:

Examples

C#

```
//--- Return AllowedConsecutiveHyphens
          int allowedHyphens = hyphenSettings.AllowedConsecutiveHyphens;

          //--- Set AllowedConsecutiveHyphens
          hyphenSettings.AllowedConsecutiveHyphens = 1;
```

vb.net

```
'--- Return AllowedConsecutiveHyphens
          Dim allowedHyphens As Boolean = hyphenSettings.AllowedConsecutiveHyphens

          '--- Set AllowedConsecutiveHyphens
          hyphenSettings.AllowedConsecutiveHyphens = 1
```

WordHyphenSettings.AutoHyphen

Description

Sets or returns a `boolean` that represents whether or not Word will automatically hyphenate this document.

Signature: C#

```
public boolean AutoHyphen{ get; set; }
```

Signature: VB.NET

```
Public Property AutoHyphen() As Boolean
```

Remarks

This option is set to false by default.

MS Word equivalent: Tools menu > Language > Hyphenation... > Automatically hyphenate document

Examples

C#

```
//--- Return AutoHyphen
          bool autoHyphen = hyphenSettings.AutoHyphen;

          //--- Set AutoHyphen
          hyphenSettings.AutoHyphen = true;
```

vb.net

```
'--- Return AutoHyphen
          Dim autoHyphen As Boolean = hyphenSettings.AutoHyphen

          '--- Set AutoHyphen
          hyphenSettings.AutoHyphen = True
```

WordHyphenSettings.CapitalHyphenation

Description

Sets or returns a `boolean` that represents whether or not Word is allowed to hyphenate words that are capitalized in this document.

Signature: C#

```
public boolean CapitalHyphenation{ get; set; }
```

Signature: VB.NET

```
Public Property CapitalHyphenation() As Boolean
```

Remarks

This option is set to true by default.

MS Word equivalent: Tools menu > Language > Hyphenation... > Hyphenate words in CAPS

Examples

C#

```
//--- Return CapitalHyphenation
          bool capitalHyphenation = hyphenSettings.CapitalHyphenation;

          //--- Set CapitalHyphenation
          hyphenSettings.CapitalHyphenation = false;
```

vb.net

```
'--- Return CapitalHyphenation
          Dim capitalHyphenation As Boolean = hyphenSettings.CapitalHyphenation

          '--- Set CapitalHyphenation
          hyphenSettings.CapitalHyphenation = False
```

WordHyphenSettings.HyphenationWidth

Description

Sets or returns an `int` that represents the hyphenation width in twips for this document. The hyphenation width is the amount of space to leave between the end of the last word in a line and the right margin. To reduce the number of hyphens, make the hyphenation zone wider. To reduce the raggedness of the right margin, make the hyphenation zone narrower. One twip = (1/20 pt) or (1/1440 in).

Signature: C#

```
public int HyphenationWidth{ get; set; }
```

Signature: VB.NET

```
Public Property HyphenationWidth() As Integer
```

Remarks

This option is set to .25 inches (360 twips) by default.

MS Word equivalent: Tools menu > Language > Hyphenation... > Hyphenation zone:

Examples

C#

```
//--- Return HyphenationWidth
          int hyphenationWidth = hyphenSettings.HyphenationWidth;

          //---- Set HyphenationWidth to .5 inches (720 twips)
          hyphenSettings.HyphenationWidth = 720;
```

vb.net

```
'--- Return HyphenationWidth
          Dim hypheationWidth As Boolean = hyphenSettings.HyphenationWidth

          '--- Set HyphenationWidth to .5 inches (720 twips)
          hyphenSettings.HyphenationWidth = 720
```

WordNumberFormat

Description

This class is used to enumerate the set of Word's built-in number formats Word. It is used for setting how List elemements, footnote numbers, and endnote numbers will appear.

Signature: C#

```
public sealed class WordNumberFormat : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class WordNumberFormat
  Inherits TypesafeEnum
```

Values

Name	Description
Arabic	Arabic Numerals
ArabicAbjad	Arabic Alphabet, Abjad sequence
ArabicAlifBaTah	Arabic Alphabet, Alif Ba Ta sequence
ArabicLeadingZero	Standard Arabic Numbers with Leading Zeros for Single Digit Values
Bullet	Simple Bullets, no numbers
CardinalText	English Cardinal Text
Chinese1	
Chinese2	
Chinese3	
Chinese4	Chinese Numerals with Parentheses; reverts to Arabic after 10
ChineseDoubleByte1	Chinese Double Byte Numerals, Additive
ChineseDoubleByte2	Chinese Double Byte Formal Numerals
ChineseDoubleByte3	Chinese Double Byte Numerals, Additive
ChineseDoubleByte4	Double Byte Arabic Numerals
ChineseZodiac1	Chinese Zodiac Elements; reverts to Arabic after 10
ChineseZodiac2	Chinese Zodiac Animals; reverts to Arabic after 12
ChineseZodiac3	Chinese Zodiac Pairs (Element, Animal); repeats after 60
Circle	
DoubleByteArabic	Double Byte Arabic Numerals

DoubleByteChar	Double Byte Arabic Numerals
DoubleByteKatakanaAiueo	Double Byte Katakana, a i u e o sequence
DoubleByteKatakanaIroha	Double Byte Katakana, i ro ha sequence
HebrewNonStandard	Hebrew Numerals
HewbrewBiblical	Hebrew Alphabetical Numbers
HindiCardinals	Hindi Cardinals
HindiConsonants	Hindi Consonants, Devanagari script
HindiNumbers	Hindi Numerals
HindiVowels	Hindi Vowels, Devanagari script
Kanji3	Japanese Formal Kanji (Daiji) Numerals
Kanji4	Japanese Kanji Numerals, Base 10
KanjiDigit	Japanese Kanji Numerals, Additive
KanjiNoDigit	Japanese Kanji Numerals, Base 10
KatakanaAiueo	Japanese Katakana Alphabet, a i u e o sequence
KatakanaIroha	Japanese Katakana Alphabet, traditional i ro ha sequence
Korean1	Korean Consonants; repeats after 14
Korean2	Korean Consonants with first Vowel; repeats after 14
KoreanDoubleByte1	Double Byte Sino-Korean
KoreanDoubleByte2	Double Byte Sino-Korean, base 10
KoreanDoubleByte3	Double Byte Native Korean
KoreanDoubleByte4	Double Byte Sino-Korean Hanja
Lowercase	Lower Case Roman Alphabet
Ordinal	English Ordinals
OrdinalText	English Ordinal Text
PageNumFormat	Arabic Numerals with Dashes
RomanUpper	Upper Case Roman Numerals
RomanLower	Lower Case Roman Numerals
RussianLower	Russian Cyrillic Alphabet Lower Case
RussianUpper	Russian Cyrillic Alphabet Upper Case
SingleByteChar	Double Byte Arabic Numerals
TaiwanDoubleByte1	Taiwanese Double Byte Numerals, Additive
TaiwanDoubleByte2	Taiwanese Double Byte Formal Numerals
TaiwanDoubleByte3	Taiwanese Double Byte Numerals, Additive
TaiwanDoubleByte4	Taiwanese Double Byte Numerals, Base 10
ThaiCardinals	Thai Cardinals
ThaiLetters	Thai Alphabet
ThaiNumbers	Thai Numerals
Uppercase	Upper Case Roman Alphabet
VietnameseCardianls	Vietnamese Cardinals

WordViewSettings

Description

This class is used manipulate the View settings for a document.

Signature: C#

```
public sealed class WordViewSettings
```

Signature: VB.NET

```
Public NotInheritable Class WordViewSettings
```

Remarks

You cannot create a new WordViewSettings object. You must use the Document.ViewSettings property to retrieve WordViewSettings properties for a document.

The following sample gets the `WordViewSettings` object of a document.

Examples

C#

```
WordApplication app = new WordApplication();
          Document doc = app.Open(@"C:\sample.doc");
          WordViewSettings viewSettings = doc.ViewSettings;
```

vb.net

```
Dim app As New WordApplication()
          Dim doc As Document = app.Open("C:\sample.doc")
          Dim viewSettings As WordViewSettings = doc.ViewSettings
```

Properties

Name	Description
DisplayedOutlineLevels	Sets or returns an `int` object that represents the number of heading levels that will be displayed by Word in the outline view for this document. This can be set to a number between 1 and 9.
DisplayFieldResults	Sets or returns a `boolean` that represents whether Word will display field results (true) or just the field text itself (false) in a document. For example, when set to true you will see the full contents of a mergefield instead of just the text value inside it.
DisplayHidden	Sets or returns a `boolean` that represents whether or not Word will display hidden contents in a document. When set to true, Word displays text formatted as hidden with a dotted underline. When you print hidden text, the dotted underline does not appear.
DocumentView	Sets or returns a WordViewSettings.ViewType object that represents the type of view this document will be displayed in when opened by Word. Possible values: Normal, Outline, Page.
EnvelopeVisible	Sets or returns a `boolean` that represents whether or not Word will display an envelope overlay with this document (if there is one).
IncludeFooterInBorder	Sets or returns a `boolean` that represents whether or not Word will display the footer inside the page border for a document.
IncludeHeaderInBorder	Sets or returns a `boolean` that represents whether or not Word will display the header inside the page border for a document.
ShadeFormFields	Sets or returns a `boolean` that represents whether or not Word will shade form fields in a document.

Zoom	Sets or returns a WordViewSettings.ZoomType object that represents the zoom type the document will be displayed with when opened by Word. Possible values: None, FullPage, PageWidth).
ZoomPercentage	Sets or returns an `int` that represents the zoom percentage this document will be displayed with when opened by Word. Possible values: Normal, Outline, Page.

Nested Classes

Name	Description
ViewType	View types (modes) in Word for a document.
ZoomType	Zoom types (modes) in Word for a document.

WordViewSettings.DisplayedOutlineLevels

Description

Sets or returns an `int` object that represents the number of heading levels that will be displayed by Word in the outline view for this document. This can be set to a number between 1 and 9.

Signature: C#

```
public int DisplayedOutlineLevels{ get; set; }
```

Signature: VB.NET

```
Public Property DisplayedOutlineLevels() As Integer
```

Remarks

This option is set to 9 (all levels) by default.

MS Word equivalent: Outlining toolbar > Show Level

Examples

C#

```
//--- Return DisplayOutlineLevels
          int displayOutlineLevels = viewSettings.DisplayOutlineLevels;

          //--- Set ZoomType
          viewSettings.DisplayOutlineLevels = 4;
```

vb.net

```
'--- Return DisplayOutlineLevels
          Dim displayOutlineLevels As Integer = viewSettings.DisplayOutlineLevels

          '--- Set DisplayOutlineLevels
          viewSettings.DisplayOutlineLevels = 4
```

WordViewSettings.DisplayFieldResults

Description

Sets or returns a `boolean` that represents whether Word will display field results (true) or just the field text itself (false) in a document. For example, when set to true you will see the full contents of a mergefield instead of just the text value inside it.

Signature: C#

```
public boolean DisplayFieldResults{ get; set; }
```

Signature: VB.NET

```
Public Property DisplayFieldResults() As Boolean
```

Remarks

This option is set to true by default.

MS Word equivalent: Tools menu > Options... > View tab > Show section > Field codes

Examples

C#

```
//--- Return DisplayFieldResults
          bool displayFieldResults = oWordViewettings.DisplayFieldResults;

          //--- Set DisplayFieldResults
          viewSettings.DisplayFieldResults = false;
```

vb.net

```
'--- Return DisplayFieldResults
          Dim displayFieldResults As Boolean = viewSettings.DisplayFieldResults

          '--- Set DisplayFieldResults
          viewSettings.DisplayFieldResults = False
```

WordViewSettings.DisplayHidden

Description

Sets or returns a `boolean` that represents whether or not Word will display hidden contents in a document. When set to true, Word displays text formatted as hidden with a dotted underline. When you print hidden text, the dotted underline does not appear.

Signature: C#

```
public boolean DisplayHidden{ get; set; }
```

Signature: VB.NET

```
Public Property DisplayHidden() As Boolean
```

Examples

C#

```
//--- Return DisplayHidden
          bool displayHidden = viewSettings.DisplayHidden;

          //--- Set DisplayHidden
          viewSettings.DisplayHidden = true;
```

vb.net

```
'--- Return DisplayHidden
          Dim displayHidden As Boolean = viewSettings.DisplayHidden

          '--- Set DisplayHidden
          viewSettings.DisplayHidden = True
```

WordViewSettings.DocumentView

Description

Sets or returns a WordViewSettings.ViewType object that represents the type of view this document will be displayed in when opened by Word. Possible values: Normal, Outline, Page.

Signature: C#

```
public ViewType DocumentView{ get; set; }
```

Signature: VB.NET

```
Public Property DocumentView() As ViewType
```

Remarks

This option is set to WordViewSettings.ViewType.Page by default.

MS Word equivalent: View menu > (Top section)

Examples

C#

```
//--- Return DocumentView
          WordViewSettings.ViewType oViewType = viewSettings.DocumentView;

          //--- Set DocumentView
          viewSettings.DocumentView = WordViewSettings.ViewType.Outline;
```

vb.net

```
'--- Return DocumentView
          Dim oViewType As WordViewSettings.ViewType = viewSettings.DocumentView

          '--- Set ShadeFormFields
          viewSettings.DocumentView = WordViewSettings.ViewType.Outline
```

WordViewSettings.EnvelopeVisible

Description

Sets or returns a `boolean` that represents whether or not Word will display an envelope overlay with this document (if there is one).

Signature: C#

```
public boolean EnvelopeVisible{ get; set; }
```

Signature: VB.NET

```
Public Property EnvelopeVisible() As Boolean
```

Remarks

This option is set to `false` by default.

Examples

C#

```
//--- Return IncludeFooterInBorder
          bool includeFooter = viewSettings.IncludeFooterInBorder;

          //--- Set IncludeFooterInBorder
          viewSettings.IncludeFooterInBorder = false;
```

vb.net

```
'--- Return IncludeFooterInBorder
          Dim includeFooter As Boolean = viewSettings.IncludeFooterInBorder

          '--- Set IncludeFooterInBorder
          viewSettings.IncludeFooterInBorder = False
```

WordViewSettings.IncludeFooterInBorder

Description

Sets or returns a `boolean` that represents whether or not Word will display the footer inside the page border for a document.

Signature: C#

```
public boolean IncludeFooterInBorder{ get; set; }
```

Signature: VB.NET

```
Public Property IncludeFooterInBorder() As Boolean
```

Remarks

This option is set to `true` by default.

MS Word equivalent: Format menu > Border and Shading... > Page Border tab > Options... button > Options section > Surround footer

Examples

C#

```
//--- Return IncludeHeaderInBorder
          bool includeHeader = viewSettings.IncludeHeaderInBorder;

          //--- Set IncludeHeaderInBorder
          viewSettings.IncludeHeaderInBorder = false;
```

vb.net

```
'--- Return IncludeHeaderInBorder
          Dim includeHeader As Boolean = viewSettings.IncludeHeaderInBorder

          '--- Set IncludeHeaderInBorder
          viewSettings.IncludeHeaderInBorder = False
```

WordViewSettings.IncludeHeaderInBorder

Description

Sets or returns a `boolean` that represents whether or not Word will display the header inside the page border for a document.

Signature: C#

```
public boolean IncludeHeaderInBorder{ get; set; }
```

Signature: VB.NET

```
Public Property IncludeHeaderInBorder() As Boolean
```

Remarks

This option is set to `true` by default.

MS Word equivalent: Format menu > Border and Shading... > Page Border tab > Options... button > Options section > Surround header

Examples

C#

```
//--- Return IncludeHeaderInBorder
          bool includeHeader = viewSettings.IncludeHeaderInBorder;

          //--- Set IncludeHeaderInBorder
          viewSettings.IncludeHeaderInBorder = false;
```

vb.net

```
'--- Return IncludeHeaderInBorder
          Dim includeHeader As Boolean = viewSettings.IncludeHeaderInBorder

          '--- Set IncludeHeaderInBorder
          viewSettings.IncludeHeaderInBorder = False
```

WordViewSettings.ShadeFormFields

Description

Sets or returns a `boolean` that represents whether or not Word will shade form fields in a document.

Signature: C#

```
public boolean ShadeFormFields{ get; set; }
```

Signature: VB.NET

```
Public Property ShadeFormFields() As Boolean
```

Remarks

This option is set to true by default.

MS Word equivalent: Tools menu > Options... > View tab > Show section > Field shading

Examples

C#

```
//--- Return ShadeFormFields
          bool shadeFormFields = viewSettings.ShadeFormFields;

          //--- Set ShadeFormFields
          viewSettings.ShadeFormFields = false;
```

vb.net

```
'--- Return ShadeFormFields
          Dim shadeFormFields As Boolean = viewSettings.ShadeFormFields

          '--- Set ShadeFormFields
          viewSettings.ShadeFormFields = False
```

WordViewSettings.ViewType

Description

View types (modes) in Word for a document.

Signature: C#

```
public sealed class ViewType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ViewType
  Inherits TypesafeEnum
```

Values

Name	Description
Normal	Document opens in default viewing settings
Outline	Document opens in Outline view
Page	Document opens in Page view

WordViewSettings.Zoom

Description

Sets or returns a WordViewSettings.ZoomType object that represents the zoom type the document will be displayed with when opened by Word. Possible values: None, FullPage, PageWidth).

Signature: C#

```
public ZoomType Zoom{ get; set; }
```

Signature: VB.NET

```
Public Property Zoom() As ZoomType
```

Remarks

This option is set to None by default.

MS Word equivalent: View menu > Zoom... > Zoom to section

Examples

C#

```
//--- Return DisplayOutlineLevels
          int displayOutlineLevels = viewSettings.DisplayOutlineLevels;

          //--- Set ZoomType
          viewSettings.DisplayOutlineLevels = 4;
```

vb.net

```
'--- Return DisplayOutlineLevels
          Dim displayOutlineLevels As Integer = viewSettings.DisplayOutlineLevels

          '--- Set DisplayOutlineLevels
          viewSettings.DisplayOutlineLevels = 4
```

WordViewSettings.ZoomPercentage

Description

Sets or returns an `int` that represents the zoom percentage this document will be displayed with when opened by Word. Possible values: Normal, Outline, Page.

Signature: C#

```
public int ZoomPercentage{ get; set; }
```

Signature: VB.NET

```
Public Property ZoomPercentage() As Integer
```

Remarks

This option is set to `100` % by default.

MS Word equivalent: View menu > Zoom... > Zoom to section

Examples

C#

```
//--- Return ZoomPercentage
          int zoomPercentage = viewSettings.ZoomPercentage;

          //--- Set ZoomPercentage
          viewSettings.ZoomPercentage = 50;
```

vb.net

```
'--- Return ZoomPercentage
          Dim zoomPercentage As Integer = viewSettings.ZoomPercentage

          '--- Set ZoomPercentage
          viewSettings.ZoomPercentage = 50
```

WordViewSettings.ZoomType

Description

Zoom types (modes) in Word for a document.

Signature: C#

```
public sealed class ZoomType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ZoomType
  Inherits TypesafeEnum
```

Values

Name	Description
FullPage	Document opens zoomed out so that the full page is visible
None	Document opens to default zoom level
PageWidth	Document opens zoomed in so that the page width matches the window width

WordWriterException

Description

WordWriterException is thrown by the Element.Search() and Element.SearchAndReplace() methods.

Signature: C#

```
[Serializable]
 public sealed class WordWriterException : System.SystemException,
System.Runtime.Serialization.ISerializable
```

Signature: VB.NET

```
<Serializable()> _
 Public NotInheritable Class WordWriterException
  Inherits System.SystemException
  Implements System.Runtime.Serialization.ISerializable
```

SharePointIntegration Programmer's Reference

Introduced in build 4.1.0.1379

SharePointIntegration

Introduced in build 4.1.0.1379

Description

The SharePointIntegration class contains the extension methods that allow users to bind SharePoint data sources to Word documents as well as open and save files in SharePoint lists and document libraries.

Signature: C#

```
public static class SharePointIntegration
```

Remarks

The SharePointIntegration class is to be used on a SharePoint server to use the WordTemplate, WordApplication, and Element objects with SharePoint. SharePointIntegration belongs to the SoftArtisans.OfficeWriter.WordWriter namespace and has the Microsoft.SharePoint class as a dependency. Additionally, the SharePointIntegration class belongs in a separate DLL, SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll, which must be added to the bin directory of your project.

Examples

SharePointIntegration is an abstract class which cannot directly be instantiated, but rather contains extension methods for the WordApplication, WordTemplate, and Element objects. Use SharePointIntegration extensions just as you would the other methods of each object, remembering to add references to the Microsoft.SharePoint namespace.

Methods

Name	Description
ImportData(Element, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataBindingProperties)	The page SharePointIntegration.ImportData(Element, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataBindingProperties) does not exist.
ImportData(Element, Microsoft.SharePoint.SPList)	Imports the data from a SharePoint List into a Table at the point of the current element in the document.
Open(WordApplication, Microsoft.SharePoint.SPDocumentLibrary, String)	Opens and returns an existing Document from a SharePoint Document Library.
Open(WordApplication, Microsoft.SharePoint.SPListItem, String)	Opens and returns an existing Document from a SharePoint List Item.
Open(WordTemplate, Microsoft.SharePoint.SPDocumentLibrary, String)	Opens the WordWriter template specified by the `templateFileName` parameter.
Open(WordTemplate, Microsoft.SharePoint.SPListItem, String)	Opens the WordWriter template specified by the `templateFileName` parameter.
Save(WordApplication, Document, Microsoft.SharePoint.SPListItem, String)	Writes a Document to a file and saves it to a SharePoint List Item.
Save(WordApplication, Document, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Writes a Document to a file and saves it to a SharePoint Document Library.
Save(WordTemplate, Microsoft.SharePoint.SPListItem, String)	Saves the generated Word file to a SharePoint List Item.

Save(WordTemplate, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Saves the generated Word file to a SharePoint Document Library.
SetDataSource(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)	Sets a main document data source to a SharePoint View. If the specified View contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Sets a main document data source to a SharePoint View. If the specified View contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(WordTemplate, Microsoft.SharePoint.SPList, String)	Sets a main document data source to a SharePoint List. If the specified List contains more than one row, WordWriter will use the first row as the data source.
SetDataSource(WordTemplate, Microsoft.SharePoint.SPList)	Sets a main document data source to a SharePoint List. If the specified List contains more than one row, WordWriter will use the first row as the data source.
SetMailMerge(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, Integer)	The page SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, Integer) does not exist.
SetMailMerge(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetMailMerge(WordTemplate, Microsoft.SharePoint.SPList, Integer)	The page SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPList, Integer) does not exist.
SetMailMerge(WordTemplate, Microsoft.SharePoint.SPList)	Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.
SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, Int)	The page SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, Int) does not exist.
SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)	Sets the specified repeat block's data source to a SharePoint View.
SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPList, String, Int)	The page SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPList, String, Int) does not exist.
SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPList, String)	Sets the specified repeat block's data source to a SharePoint List.

SharePointIntegration.ImportData(Element, Microsoft.SharePoint.SPList)

Introduced in build 4.1.0.1379

Description

Imports the data from a SharePoint List into a Table at the point of the current element in the document.

Signature: C#

```
public static Table ImportData(this Element element, Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal element As Element, ByVal list As Microsoft.SharePoint.SPList)
As Table
```

Parameters

element

The current element which is calling ImportData

list

The SharePoint List to be used as a data source

Returns

The `Table` into which the data is being imported

Remarks

> This is an extension method for the Element object to be used for binding SharePoint data to documents from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
Table tbl = e.ImportData(myList);
```

vb.net

```
Dim tbl As Table = e.ImportData(myList)
```

SharePointIntegration.ImportData(Element, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)

Introduced in build 4.1.0.1379

Description

Imports the data from a SharePoint View into a Table at the point of the current element in the document. Also accepts a SharePoint List to which the View belongs and a DataImportProperties object with the import options.

Signature: C#

```
public static Table ImportData(this Element element, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, DataImportProperties props)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal element As Element, ByVal view As Microsoft.SharePoint.SPView,
ByVal list As Microsoft.SharePoint.SPList, ByVal props As DataImportProperties) As Table
```

Parameters

element

The current element which is calling ImportData

view

The SharePoint View to be used as a data source

list

SharePoint List to which the View belongs

props

Import properties for this data import, including max rows, max columns, column filter, etc.

Returns

The `Table` into which the data is being imported

Remarks

> This is an extension method for the Element object to be used for binding SharePoint data to documents from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
Table tbl = e.ImportData(myView, myList, importProps);
```

vb.net

```
Dim tbl As Table = e.ImportData(myView, myList, importProps)
```

SharePointIntegration.Open(WordApplication, Microsoft.SharePoint.SPDocumentLibrary, String)

Introduced in build 4.1.0.1379

Description

Opens and returns an existing Document from a SharePoint Document Library.

Signature: C#

```
public Document Open(this WordApplication wordApplication, Microsoft.SharePoint.SPDocumentLibrary
docLib, String fileName)
```

Signature: VB.NET

```
Public Function Open(ByVal wordApplication As WordApplication, ByVal docLib As
Microsoft.SharePoint.SPDocumentLibrary, ByVal fileName As String) As Document
```

Parameters

wordApplication

The current WordApplication object which is calling Open

docLib

SharePoint Document Library from which the file is being opened

fileName

Filename of the document to open from the Document Library

Returns

The document that is opened from the Document Library.

Exceptions

System.Exception

Thrown if an error occurs while opening the document.

Remarks

Though the WordTemplate object currently supports Office Open XML (Word 2007) format files, WordApplication.Open() does not yet support them.

> This is an extension method for the WordApplication object to be used for opening documents from a SharePoint Document Library from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
Document doc = app.Open(myDocLib, "sample.doc");
```

vb.net

```
Dim doc As Document = app.Open(myDocLib, "sample.doc")
```

SharePointIntegration.Open(WordApplication, Microsoft.SharePoint.SPListItem, String)

Introduced in build 4.1.0.1379

Description

Opens and returns an existing Document from a SharePoint List Item.

Signature: C#

```
public Document Open(this WordApplication wordApplication, Microsoft.SharePoint.SPListItem listItem,
String fileName)
```

Signature: VB.NET

```
Public Function Open(ByVal wordApplication As WordApplication, ByVal listItem As
Microsoft.SharePoint.SPListItem, ByVal fileName As String) As Document
```

Parameters

wordApplication

The current WordApplication object which is calling Open

listItem

SharePoint List Item from which the file is being opened

fileName

Filename of the document to open from the List Item

Returns

The document that is opened from the List Item.

Exceptions

System.Exception

Thrown if an error occurs while opening the document.

Remarks

Though the WordTemplate object currently supports Office Open XML (Word 2007) format files, WordApplication.Open() does not yet support them.

This is an extension method for the WordApplication object to be used for opening documents from a SharePoint List Item from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
Document doc = app.Open(myListItem, "sample.doc");
```

vb.net

```
Dim doc As Document = app.Open(myListItem, "sample.doc")
```

SharePointIntegration.Open(WordTemplate, Microsoft.SharePoint.SPDocumentLibrary, String)

Introduced in build 4.1.0.1379

Description

Opens the WordWriter template specified by the `templateFileName` parameter.

Signature: C#

```
public static void Open(this WordTemplate template, Microsoft.SharePoint.SPDocumentLibrary docLib,
String fileName)
```

Signature: VB.NET

```
Public Static Sub Open(ByVal template As WordTemplate, ByVal docLib As
MicroSoft.SharePoint.SPDocumentLibrary, ByVal fileName As String)
```

Parameters

template

The current WordTemplate that is calling Open

docLib

SharePoint Document Library containing the template file to be opened

fileName

File name of the template Word file

Exceptions

ArgumentNullException FileNotFoundException

Remarks

A WordWriter template is a file created in Microsoft Word that contains merge fields. WordWriter supports Microsoft Word 97, 2000, 2002 (XP), 2003, and 2007. Do not use Open to open files created in earlier versions of Microsoft Word.

WordWriter templates can be created in Windows 2003, however, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.

This is an extension method for the WordTemplate object to be used for opening documents from a SharePoint Document Library from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
oWW.Open(myDocLib, "StringTemplate.doc");
```

vb.net

```
oWW.Open(myDocLib, "StringTemplate.doc")
```

SharePointIntegration.Open(WordTemplate, Microsoft.SharePoint.SPListItem, String)

Introduced in build 4.1.0.1379

Description

Opens the WordWriter template specified by the `templateFileName` parameter.

Signature: C#

```
public static void Open(this WordTemplate template, Microsoft.SharePoint.SPListItem listItem, String
fileName)
```

Signature: VB.NET

```
Public Static Sub Open(ByVal template As WordTemplate, ByVal listItem As
MicroSoft.SharePoint.SPListItem, ByVal fileName As String)
```

Parameters

template

The current WordTemplate that is calling Open

listItem

SharePoint List Item containing the template file to be opened

fileName

File name of the template Word file

Exceptions

ArgumentNullException FileNotFoundException

Remarks

A WordWriter template is a file created in Microsoft Word that contains merge fields. WordWriter supports Microsoft Word 97, 2000, 2002 (XP), 2003, and 2007. Do not use Open to open files created in earlier versions of Microsoft Word.

WordWriter templates can be created in Windows 2003, however, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.

This is an extension method for the WordTemplate object to be used for opening documents from a SharePoint List Item from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
oWW.Open(myListItem, "StringTemplate.doc");
```

vb.net

```
oWW.Open(myListItem, "StringTemplate.doc")
```

SharePointIntegration.Save(WordApplication, Document, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)

Introduced in build 4.1.0.1379

Description

Writes a Document to a file and saves it to a SharePoint Document Library.

Signature: C#

```
public static void Save(this WordApplication wordApplication, Document document,
Microsoft.SharePoint.SPDocumentLibrary docLib, string fileName, bool overwrite)
```

Signature: VB.NET

```
Public Shared Sub Save(ByVal wordApplication As WordApplication, ByVal document As Document, ByVal
docLib As Microsoft.SharePoint.SPDocumentLibrary, ByVal fileName As String, ByVal overwrite As
Boolean)
```

Parameters

wordApplication

The current WordApplication object from which Save is being called

document

The `Document` to save.

docLib

The SharePoint Document Library to which the file will be saved

fileName

Filename of the `Document`

overwrite

If true, will overwrite an existing file of the same name in the Document Library

Exceptions

System.Exception

Thrown if there is an error saving the file.

Remarks

The WordApplication.Preserve property is used to specify the way WordWriter will save the document. When the `preserve` property is set to true, WordWriter attempts to preserve all formatting and features that it doesn't directly support. This includes Fields, Footnotes, Hyperlinks, Comments, and Anchored Images. When the `preserve` property is false, only those features directly supported by WordWriter will be preserved. This includes all Tables, Lists, Headers and Footers.

> This is an extension method for the WordApplication object to be used for saving documents to a SharePoint Document Library from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Retrieve current AllowUnsafeUpdates value
bool currentAllowUnsafeUpdates = documentLibrary.ParentWeb.AllowUnsafeUpdates;

//--- You must allow unsafe updates in order for the file to be saved in the
//--- Document Library from a web application
documentLibrary.ParentWeb.AllowUnsafeUpdates = true;
wa.Save(doc, documentLibrary, "populated.doc", true);

//--- Update the document library
documentLibrary.Update();

//--- Return the AllowUnsafeUpdates value to its original state
documentLibrary.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates;
```

vb.net

```
'--- Retrieve current AllowUnsafeUpdates value
Dim currentAllowUnsafeUpdates As Boolean = documentLibrary.ParentWeb.AllowUnsafeUpdates

'--- You must allow unsafe updates in order for the file to be saved in the
'--- Document Library from a web application
documentLibrary.ParentWeb.AllowUnsafeUpdates = True
wa.Save(doc, documentLibrary, "populated.doc", True)

'--- Update the document library
documentLibrary.Update()

'--- Return the AllowUnsafeUpdates value to its original state
documentLibrary.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates
```

SharePointIntegration.Save(WordApplication, Document, Microsoft.SharePoint.SPListItem, String)

Introduced in build 4.5

Description

Writes a Document to a file and saves it to a SharePoint List Item.

Signature: C#

```
public static void Save(this WordApplication wordApplication, Document document,
Microsoft.SharePoint.SPListItem listItem, string fileName)
```

Signature: VB.NET

```
Public Shared Sub Save(ByVal wordApplication As WordApplication, ByVal document As Document, ByVal
docLib As Microsoft.SharePoint.SPListItem, ByVal fileName As String)
```

Parameters

wordApplication

The current WordApplication object from which Save is being called

document

The `Document` to save.

listItem

The SharePoint List Item to which the file will be saved

fileName

Filename of the `Document`

Exceptions

System.Exception

Thrown if there is an error saving the file.

Remarks

The WordApplication.Preserve property is used to specify the way WordWriter will save the document. When the `preserve` property is set to true, WordWriter attempts to preserve all formatting and features that it doesn't directly support. This includes Fields, Footnotes, Hyperlinks, Comments, and Anchored Images. When the `preserve` property is false, only those features directly supported by WordWriter will be preserved. This includes all Tables, Lists, Headers and Footers.

> This is an extension method for the WordApplication object to be used for saving documents to a SharePoint List Item from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Retrieve current AllowUnsafeUpdates value
bool currentAllowUnsafeUpdates = list.ParentWeb.AllowUnsafeUpdates;

//--- You must allow unsafe updates in order for the file to be saved in the
//--- List from a web application
list.ParentWeb.AllowUnsafeUpdates = true;
wa.Save(doc, listItem, "populated.doc");

//--- Update the list
listItem.Update();

//--- Return the AllowUnsafeUpdates value to its original state
list.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates;
```

vb.net

```
'--- Retrieve current AllowUnsafeUpdates value
Dim currentAllowUnsafeUpdates As Boolean = list.ParentWeb.AllowUnsafeUpdates

'--- You must allow unsafe updates in order for the file to be saved in the
'--- List from a web application
list.ParentWeb.AllowUnsafeUpdates = True
wa.Save(doc, listItem, "populated.doc")

'--- Update the list
listItem.Update()

'--- Return the AllowUnsafeUpdates value to its original state
list.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates
```

SharePointIntegration.Save(WordTemplate, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)

Introduced in build 4.1.0.1379

Description

Saves the generated Word file to a SharePoint Document Library.

Signature: C#

```
public static void Save(this WordTemplate template, Microsoft.SharePoint.DocumentLibrary docLib,
string fileName, bool overwrite)
```

Signature: VB.NET

```
Public Shared Sub Save(ByVal template As WordTemplate, ByVal docLib As
Microsoft.SharePoint.DocumentLibrary, ByVal fileName As String, ByVal overwrite As Boolean)
```

Parameters

template

The current WordTemplate object that is being saved to file

docLib

SharePoint document library to which the file is being saved

fileName

Name of the file to be saved

overwrite

If set to true, will overwrite any files in the Document Library by that name

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

Remarks

You can call Save more than once for a single instance of WordTemplate. This allows you to save more than one copy of a generated file, and/or both save the file on the server and stream it to the client.

This is an extension method for the WordTemplate object to be used for saving documents to a SharePoint Document Library from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Retrieve current AllowUnsafeUpdates value
bool currentAllowUnsafeUpdates = documentLibrary.ParentWeb.AllowUnsafeUpdates;

//--- You must allow unsafe updates in order for the file to be saved in the
//--- Document Library from a web application
documentLibrary.ParentWeb.AllowUnsafeUpdates = true;
wt.Save(documentLibrary, "populated.docx", true);

//--- Update the document library
documentLibrary.Update();

//--- Return the AllowUnsafeUpdates value to its original state
documentLibrary.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates;
```

vb.net

```
'--- Retrieve current AllowUnsafeUpdates value
Dim currentAllowUnsafeUpdates As Boolean = documentLibrary.ParentWeb.AllowUnsafeUpdates

'--- You must allow unsafe updates in order for the file to be saved in the
'--- Document Library from a web application
documentLibrary.ParentWeb.AllowUnsafeUpdates = True
wt.Save(documentLibrary, "populated.docx", True)

'--- Update the document library
documentLibrary.Update()

'--- Return the AllowUnsafeUpdates value to its original state
documentLibrary.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates
```

SharePointIntegration.Save(WordTemplate, Microsoft.SharePoint.SPListItem, String)

Introduced in build 4.1.0.1379

Description

Saves the generated Word file to a SharePoint List Item.

Signature: C#

```
public static void Save(this WordTemplate template, Microsoft.SharePoint.ListItem listItem, string
fileName)
```

Signature: VB.NET

```
Public Shared Sub Save(ByVal template As WordTemplate, ByVal listItem As
Microsoft.SharePoint.ListItem, ByVal fileName As String)
```

Parameters

template

The current WordTemplate object that is being saved to file

listItem

SharePoint List Item to which the file is being saved

fileName

Name of the file to be saved

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

Remarks

You can call Save more than once for a single instance of WordTemplate. This allows you to save more than one copy of a generated file, and/or both save the file on the server and stream it to the client.

> This is an extension method for the WordTemplate object to be used for saving documents to a SharePoint List Item from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Retrieve current AllowUnsafeUpdates value
bool currentAllowUnsafeUpdates = list.ParentWeb.AllowUnsafeUpdates;

//--- You must allow unsafe updates in order for the file to be saved in the
//--- List from a web application
list.ParentWeb.AllowUnsafeUpdates = true;
wt.Save(listItem, "populated.docx");

//--- Update the list
listItem.Update();

//--- Return the AllowUnsafeUpdates value to its original state
list.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates;
```

vb.net

```
'--- Retrieve current AllowUnsafeUpdates value
Dim currentAllowUnsafeUpdates As Boolean = list.ParentWeb.AllowUnsafeUpdates

'--- You must allow unsafe updates in order for the file to be saved in the
'--- List from a web application
list.ParentWeb.AllowUnsafeUpdates = True
wt.Save(listItem, "populated.docx")

'--- Update the list
listItem.Update()

'--- Return the AllowUnsafeUpdates value to its original state
list.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates
```

SharePointIntegration.SetDataSource(WordTemplate, Microsoft.SharePoint.SPList, String)

Introduced in build 4.1.0.1379

Description

Sets a main document data source to a SharePoint List. If the specified List contains more than one row, WordWriter will use the first row as the data source.

Signature: C#

```
public static void SetDataSource(this WordTemplate template, Microsoft.SharePoint.SPList list, string name)
```

Signature: VB.NET

```
Public Shared Sub SetDataSource(ByVal template As WordTemplate, ByVal list As Microsoft.SharePoint.SPList, ByVal name As String)
```

Parameters

template

The current ExcelTemplate which is calling SetDataSource

list

A SharePoint List to use as the data source. WordWriter will use the first row of the List as the data source.

name

Specifies the data source name for the set of merge fields.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

This method may be called once for each set of merge fields in the main document.

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Pass the SPList to SetDataSource.
          oWW.SetDataSource(myList, "Employee");
```

vb.net

```
'--- Pass the SPList to SetDataSource.
          oWW.SetDataSource(myList, "Employee")
```

SharePointIntegration.SetDataSource(WordTemplate, Microsoft.SharePoint.SPList)

Introduced in build 4.1.0.1379

Description

Sets a main document data source to a SharePoint List. If the specified List contains more than one row, WordWriter will use the first row as the data source.

Signature: C#

```
public static void SetDataSource(this WordTemplate template, Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Shared Sub SetDataSource(ByVal template As WordTemplate, ByVal list As
Microsoft.SharePoint.SPList)
```

Parameters

template

The current ExcelTemplate which is calling SetDataSource

list

A SharePoint List to use as the data source. WordWriter will use the first row of the List as the data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

Do not call this method more than once for a single instance of WordTemplate. To set multiple main document data sources, use the following methods:
SetDataSource(Object(), String(), String)
SetDataSource(System.Data.DataSet, String)
SetDataSource(System.Data.DataTable, String)
SetDataSource(System.Data.IDataReader, String)
SetDataSource(Microsoft.SharePoint.SPList, String)
SetDataSource(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Pass the SPList to SetDataSource.
          oWW.SetDataSource(myList);
```

vb.net

```
'--- Pass the SPList to SetDataSource.
          oWW.SetDataSource(myList)
```

SharePointIntegration.SetDataSource(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)

Introduced in build 4.1.0.1379

Description

Sets a main document data source to a SharePoint View. If the specified View contains more than one row, WordWriter will use the first row as the data source.

Signature: C#

```
public static void SetDataSource(this WordTemplate template, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, string name)
```

Signature: VB.NET

```
Public Shared Sub SetDataSource(ByVal template As WordTemplate, ByVal view As
Microsoft.SharePoint.SPView, ByVal list As Microsoft.SharePoint.SPList, ByVal name As String)
```

Parameters

template

The current ExcelTemplate which is calling SetDataSource

view

A SharePoint View to use as the data source. WordWriter will use the first row of the View as the data source

list

A SharePoint List to which the View applies

name

Specifies the data source name for the set of merge fields

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

This method may be called once for each set of merge fields in the main document.

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Pass the SPList to SetDataSource.
          oWW.SetDataSource(myView, myList, "Employee");
```

vb.net

```
'--- Pass the SPList to SetDataSource.
          oWW.SetDataSource(myView, myList, "Employee")
```

SharePointIntegration.SetDataSource(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)

Introduced in build 4.1.0.1379

Description

Sets a main document data source to a SharePoint View. If the specified View contains more than one row, WordWriter will use the first row as the data source.

Signature: C#

```
public static void SetDataSource(this WordTemplate template, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Shared Sub SetDataSource(ByVal template As WordTemplate, ByVal view As
Microsoft.SharePoint.SPView, ByVal list As Microsoft.SharePoint.SPList)
```

Parameters

template

The current ExcelTemplate which is calling SetDataSource

view

A SharePoint View to use as the data source. WordWriter will use the first row of the View as the data source

list

A SharePoint List to which the View applies

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Each merge field in a WordWriter template must bind to a data source field/value pair. The number of merge fields in the template may not exceed the number of values in the data source. However, the number of values in the data source may be greater than the number of merge fields in the template.

Do not call this method more than once for a single instance of WordTemplate. To set multiple main document data sources, use the following methods:
SetDataSource(Object(), String(), String)
SetDataSource(System.Data.DataSet, String)
SetDataSource(System.Data.DataTable, String)
SetDataSource(System.Data.IDataReader, String)
SetDataSource(Microsoft.SharePoint.SPList, String)
SetDataSource(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Pass the SPList to SetDataSource.
          oWW.SetDataSource(myView, myList);
```

vb.net

```
'--- Pass the SPList to SetDataSource.
          oWW.SetDataSource(myView, myList)
```

SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPList, Int32)

Introduced in build 4.1.0.1379

Description

Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(this WordTemplate template, Microsoft.SharePoint.SPList list, int maxRows)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal template As WordTemplate, ByVal list As Microsoft.SharePoint.SPList,
ByVal maxRows As Integer)
```

Parameters

template

Current WordTemplate that is calling SetMailMerge

list

A SharePoint List to use as a data source.

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetDataSource include field names (**«fieldname»**) and field ordinals (**«#1»**).

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Set the mail merge
          //--- The data source is the SharePoint List myList
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(myList, WordTemplate.ALL_ROWS);
          wt.Process();
```

vb.net

```
'--- Set the mail merge
          '--- The data source is the SharePoint List myList
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(myList, WordTemplate.ALL_ROWS)
          wt.Process()
```

SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPList)

Introduced in build 4.1.0.1379

Description

Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(this WordTemplate template, Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal template As WordTemplate, ByVal list As Microsoft.SharePoint.SPList)
```

Parameters

template

Current WordTemplate that is calling SetMailMerge

list

A SharePoint List to use as a data source.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetDataSource include field names (**«fieldname»**) and field ordinals (**«#1»**).

This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Set the mail merge
          //--- The data source is the SharePoint List myList
          wt.SetMailMerge(myList);
          wt.Process();
```

vb.net

```
'--- Set the mail merge
          '--- The data source is the SharePoint List myList
          wt.SetMailMerge(myList)
          wt.Process()
```

SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, Int32)

Introduced in build 4.1.0.1379

Description

Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(this WordTemplate template, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, int maxRows)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal template As WordTemplate, ByVal view As Microsoft.SharePoint.SPView,
ByVal list As Microsoft.SharePoint.SPList, ByVal maxRows As Integer)
```

Parameters

template

Current WordTemplate that is calling SetMailMerge

view

A SharePoint View to use as a data source.

list

A SharePoint List to which the view applies

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetDataSource include field names (**«fieldname»**) and field ordinals (**«#1»**).

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Set the mail merge
          //--- The data source is the SharePoint View myView
          //--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(myView, myList, WordTemplate.ALL_ROWS);
          wt.Process();
```

vb.net

```
'--- Set the mail merge
          '--- The data source is the SharePoint View myView
          '--- MaxRows is set to ALL_ROWS, which allows all rows to be imported
          wt.SetMailMerge(myView, myList, WordTemplate.ALL_ROWS)
          wt.Process()
```

SharePointIntegration.SetMailMerge(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)

Introduced in build 4.1.0.1379

Description

Sets the MailMerge's data source to a SharePoint View. A MailMerge by default treats the page content as a repeat block, repeating the page content for each row unless NEXT fields are used. If the WordTemplate.EnableNEXTFields property is set to true, the NEXT field can also be used to indicate that the next row of data should be inserted instead of the current row at the next occurence of the merge fields.

Signature: C#

```
public void SetMailMerge(this WordTemplate template, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Sub SetMailMerge(ByVal template As WordTemplate, ByVal view As Microsoft.SharePoint.SPView,
ByVal list As Microsoft.SharePoint.SPList)
```

Parameters

template

Current WordTemplate that is calling SetMailMerge

view

A SharePoint View to use as a data source.

list

A SharePoint List to which the view applies

Exceptions

ArgumentNullException

Save will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can call SetMailMerge once for each instance of WordTemplate. If you are using the Excel 2003 binary template file type (.xls/.xlt), you can call SetMailMerge or SetRepeatBlock, but not both.

Additionally, only the page content is repeated for each row, not the entire page itself. If you wish to have the page itself repeat for each row, you will need to remember to place a page break at the bottom of the page.

Merge Fields for using the SetMailMerge method must not specify a data source – the data source is implied, and using a data source name will cause WordTemplate to throw an error. Valid merge field formats for use with SetDataSource include field names (**«fieldname»**) and field ordinals (**«#1»**).

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Set the mail merge
          //--- The data source is the SharePoint View myView
          wt.SetMailMerge(myView, myList);
          wt.Process();
```

vb.net

```
'--- Set the mail merge
          '--- The data source is the SharePoint View myView
          wt.SetMailMerge(myView, myList)
          wt.Process()
```

SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPList, String, Int32)

Introduced in build 4.1.0.1379

Description

Sets the specified repeat block's data source to a SharePoint List.

Signature: C#

```
public static void SetRepeatBlock(this WordTemplate template, Microsoft.SharePoint.SPList list,
System.String bookmark, int maxRows)
```

Signature: VB.NET

```
Public Shared Sub SetRepeatBlock(ByVal template As WordTemplate, ByVal list As
Microsoft.SharePoint.SPList, ByVal bookmark As String, ByVal maxRows As Integer)
```

Parameters

template

The current WordTemplate object calling SetRepeatBlock

list

The SharePoint List to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

maxRows

Specifies the maximum number of repetitions to write to the file. If Next fields are enabled (see EnableNEXTFields), then the number of rows imported from the data source is `maxRows*(numberOfNextFields+1)`. If Next fields are not enabled the number of rows imported is equal to the number of repetitions written out. To write the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "myList" is the SharePoint List containing the data to be
          //--- imported and repeated in the repeat block
          //--- Set MaxRows to 10.  Only 10 rows from the SharePoint List
          //--- will be imported
          oWW.SetRepeatBlock(myList, "Page", 10);
          oWW.Process();
```

vb.net

```
'--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "myList" is the SharePoint List containing the data to be
          '--- imported and repeated in the repeat block
          '--- Set MaxRows to 10.  Only 10 rows from the SharePoint List
          '--- will be imported
          oWW.SetRepeatBlock(myList, "Page", 10)
          oWW.Process()
```

SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPList, String)

Introduced in build 4.1.0.1379

Description

Sets the specified repeat block's data source to a SharePoint List.

Signature: C#

```
public static void SetRepeatBlock(this WordTemplate template, Microsoft.SharePoint.SPList list,
System.String bookmark)
```

Signature: VB.NET

```
Public Shared Sub SetRepeatBlock(ByVal template As WordTemplate, ByVal list As
Microsoft.SharePoint.SPList, ByVal bookmark As String)
```

Parameters

template

The current WordTemplate object calling SetRepeatBlock

list

The SharePoint List to use as the data source.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "myList" is the SharePoint List containing the data to be
          //--- imported and repeated in the repeat block
          oWW.SetRepeatBlock(myList, "Page");
          oWW.Process();
```

vb.net

```
'--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "myList" is the SharePoint List containing the data to be
          '--- imported and repeated in the repeat block
          oWW.SetRepeatBlock(myList, "Page")
          oWW.Process()
```

SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, Int32)

Introduced in build 4.1.0.1379

Description

Sets the specified repeat block's data source to a SharePoint View.

Signature: C#

```
public static void SetRepeatBlock(this WordTemplate template, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, string bookmark, int maxRows)
```

Signature: VB.NET

```
Public Shared Sub SetRepeatBlock(ByVal template As WordTemplate, ByVal view As
Microsoft.SharePoint.SPView, ByVal list As Microsoft.SharePoint.SPList, ByVal bookmark As String,
ByVal maxRows As Integer)
```

Parameters

template

The current WordTemplate object calling SetRepeatBlock

view

The SharePoint View to use as the data source.

list

The SharePoint List to which the View applies.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.

2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

maxRows

Specifies the maximum number of repetitions to write to the file. If Next fields are enabled (see EnableNEXTFields), then the number of rows imported from the data source is `maxRows*(numberOfNextFields+1)`. If Next fields are not enabled the number of rows imported is equal to the number of repetitions written out. To write the maximum rows available, use the constant WordTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "myView" is the SharePoint View containing the data to be
          //--- imported and repeated in the repeat block
          //--- Set MaxRows to 10.  Only 10 rows from the SharePoint List
          //--- will be imported
          oWW.SetRepeatBlock(myView, myList, "Page", 10);
          oWW.Process();
```

vb.net

```
'--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "myView" is the SharePoint View containing the data to be
          '--- imported and repeated in the repeat block
          '--- Set MaxRows to 10.  Only 10 rows from the SharePoint List
          '--- will be imported
          oWW.SetRepeatBlock(myView, myList, "Page", 10)
          oWW.Process()
```

SharePointIntegration.SetRepeatBlock(WordTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String)

Introduced in build 4.1.0.1379

Description

Sets the specified repeat block's data source to a SharePoint View.

Signature: C#

```
public static void SetRepeatBlock(this WordTemplate template, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, string bookmark)
```

Signature: VB.NET

```
Public Shared Sub SetRepeatBlock(ByVal template As WordTemplate, ByVal view As
Microsoft.SharePoint.SPView, ByVal list As Microsoft.SharePoint.SPList, ByVal bookmark As String)
```

Parameters

template

The current WordTemplate object calling SetRepeatBlock

view

The SharePoint View to use as the data source.

list

The SharePoint List to which the View applies.

bookmark

The bookmark name of the template repeat block. The bookmark passed to `SetRepeatBlock` must exist in the template Word file. To see a list of template bookmark names:

1. Open the template in Microsoft Word.
2. Open the **Edit** menu.
3. Select **Go To...**
4. Select **Bookmark**.
5. Under **Enter bookmark name**, open the drop-down list of bookmark names.

Exceptions

ArgumentNullException

`SetRepeatBlock` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A repeat block is a fragment in the template document that will be repeated for each row in a data source. In the template document, repeat blocks are defined by Word bookmarks that contain merge fields.

You can call SetRepeatBlock several times for a single instance of WordTemplate. The repeat block specified by the parameter `bookmark` must exist in the template.

> This is an extension method for the WordTemplate object to be used for binding SharePoint data to applications from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.WordWriter.SharePointIntegration.dll

Examples

C#

```
//--- Set a data source for the repeat block
          //--- defined by the bookmark "Page"
          //--- "myView" is the SharePoint View containing the data to be
          //--- imported and repeated in the repeat block
          oWW.SetRepeatBlock(myView, myList, "Page");
          oWW.Process();
```

vb.net

```
'--- Set a data source for the repeat block
          '--- defined by the bookmark "Page"
          '--- "myView" is the SharePoint View containing the data to be
          '--- imported and repeated in the repeat block
          oWW.SetRepeatBlock(myView, myList, "Page")
          oWW.Process()
```

Installation

- Installing WordWriter
- ASP Installation
- Removing WordWriter
- Security Considerations

Installing WordWriter

- Upgrading to a New Version of WordWriter
- Automatic Installation
- Installing WordWriter at the Application Level
- Installing WordWriter in the Global Assembly Cache (GAC)
- Using WordWriter with Windows 2003
- ASP Installation

Upgrading to a New Version of WordWriter

If you are upgrading to a new version of WordWriter before installing the new version, remove the old version.

Automatic Installation

To run the automatic installation, double-click OfficeWriter-Installer-4.x.x.exe and follow the instructions. By default, the automatic installation will save WordWriter in the directory C:\Program Files\SoftArtisans\OfficeWriter. You can select a different directory during the installation process.

WordWriter includes the .Net assembly SoftArtisans.OfficeWriter.WordWriter.dll
The .Net assembly is installed in:

1. **OfficeWriter\samples\bin**

   This is the bin directory for WordWriters samples.

During the automatic installation process you will be asked to enter the WordWriter license key that you received with your order confirmation. To see all OfficeWriter license keys installed in the registry, or to add or remove a key (for example, when upgrading) run LicenseManager.exe (included with WordWriter).

There is one installer for OfficeWriter. Depending on which license key is entered during installation (or updated later with the license utility), different functionality will be unlocked for WordWriter. For information on the different WordWriter versions, see WordWriter Editions.
If you leave the key field blank during installation you will get WordWriterFree, an evaluation edition of WordWriter. If you enter a key for an evaluation edition, you will get an evaluation edition of WordWriterEE.
When you install WordWriter, you will also be given the option to install ExcelWriter. If your license key is only for WordWriter, and you choose to install ExcelWriter as well, a free limited version of ExcelWriter will be installed. If you ordered an evaluation or full version of the whole OfficeWriter suite, you will receive one OfficeWriter key which will unlock both products.

When you update a license key, IIS must be reset before the new license will take effect.

Installing WordWriter at the Application Level

If you removed the SoftArtisans.OfficeWriter.WordWriter.dll from the GAC, it will only be available to the WordWriter samples. To make WordWriter available to another ASP.NET application, place the assembly in the **bin** directory at the top level of the application:

1. At the top level of your application, create a directory called **bin**.
2. Copy SoftArtisans.OfficeWriter.WordWriter.dll from OfficeWriter\bin\dotnet20 to your application's bin directory.

Installing WordWriter in the Global Assembly Cache (GAC)

To install SoftArtisans.OfficeWriter.WordWriter.dll in the GAC:

1. Open a command prompt window and move to the directory OfficeWriter\bin\dotnet20.
2. Enter gacutil /i SoftArtisans.OfficeWriter.WordWriter.dll.
3. Open the file **machine.config** (in [Windows directory]\Microsoft.NET\Framework[.NET version directory]\CONFIG).
4. Add the following line to the **assemblies** node of machine.config:

```
<add assembly="SoftArtisans.OfficeWriter.WordWriter, Version=x.x.x.x, Culture=neutral,
  PublicKeyToken=f593502af6ee46ae"/>
```

OR
Create a text file containing the following lines, and save it as **web.config**. Save web.config at the top level of your application.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.web>
      <compilation>
         <compilers>
         </compilers>
         <assemblies>
            <add assembly="SoftArtisans.OfficeWriter.WordWriter, Version=x.x.x.x,
Culture=neutral,
               PublicKeyToken=f593502af6ee46ae"/>
            <add assembly="*" />
         </assemblies>
      </compilation>
   </system.web>
</configuration>
```

The version attribute of the add assembly node must correspond **exactly** to the version of the assembly (dll file) added to the GAC. If you add a new version of WordWriter to the GAC using the gacutil command, update the dll versionattributes in machine.config. To get the exact version of the dll file, right-click the dll and select the Properties tab.

Using WordWriter with Windows 2003

To use WordWriter on Windows 2003, you must enable the following Web Service extensions in IIS:

- Active Server Pages
- ASP.NET

To enable these extensions:

1. Open the Internet Information Services (IIS) Manager.
2. Open **local computer**.
3. Select **Web Service Extensions**.
4. From the list of Web Service Extensions, select **Active Server Pages**.
5. Click the **Allow** button.
6. From the list of Web Service Extensions, select **ASP.NET**.
7. Click the **Allow** button.

ASP Installation

Notes:

- The WordWriter COM-callable wrapper is no longer included in the standard install for OfficeWriter. However it is available for downloading in a separate package. Please contact SoftArtisans directly for information.
- The CCW (WordTemplateCCW.dll) is a COM object that makes internal calls to the WordWriter .NET assembly. Therefore, to use WordWriter in ASP, all of the system requirements for running WordWriter in ASP.NET must be satisfied.
- The WordWriter .NET assembly is compiled with /anycpu and can run natively on 64-bit operating systems. However the CCW dll is 32-bit and must run in a 32-bit process. Make sure you assign your WordWriter classic ASP virtual application to an application pool that is set to run in 32-bit compatibility mode

SoftArtisans.OfficeWriter.WordWriter.dll is a .NET assembly. A COM callable wrapper (CCW), WordTemplateCCW.dll that allows you to use the .NET WordTemplate object from ASP is available upon request. The assembly must be installed in the Global Assembly Cache (GAC) to allow the CCW to work. Therefore you must install the WordWriter .NET assembly in the GAC to use WordWriter in ASP.

1. **Confirm that the Microsoft Visual C++ 2005 SP1 Redistributable is installed**
   This was not required in previous versions, but because of changes in the way that Microsoft handles COM-callable wrappers, this step is required to install WordWriter 4 for ASP. If you are installing WordWriter on a 64-bit machine, make sure to install the 32-bit version of the Visual C++ 2005 SP1 Redistributable.

   Download Visual C++ 2005 SP1 Redistributable for x86 systems

2. **Install SoftArtisans.OfficeWriter.WordWriter.dll in the Global Assembly Cache:**

   a. Copy SoftArtisans.OfficeWriter.WordWriter.dll to C:\WINDOWS\Assembly.
   b. On Windows 7 or Windows 2008, use gacutil.exe.
      - Unless you have Visual Studio installed on the machine (which most servers do not), you may need to download the Windows SDK to get a copy of gacutil.exe:
      - Open a command prompt and navigate to the directory with the .NET dll and enter:

```
gacutil /i SoftArtisans.OfficeWriter.WordWriter.dll
```

3. **Register WordTemplateCCW.dll:**

   a. Open a command prompt window and move to the directory OfficeWriter\WordWriter\bin\dotnet10.
   b. Enter the following:

```
regsvr32 WordTemplateCCW.dll
```

   A dialog box will inform you that WordTemplateCCW.dll registered successfully; click Ok.
      - On a 64-bit version make sure to use the 32-bit version of regsvr32, usually located in C:\Windows\SysWOW64. So you would enter:

```
C:\Windows\SysWOW64\regsvr32 WordTemplateCCW.dll
```

   c. Enter iisreset to reset IIS.

Removing WordWriter

Automatic Removal

If WordWriter was installed with the automatic installer, use **Add/Remove Programs** to remove WordWriter and all of its associated documentation and samples:

1. From the **Start** menu, select **Settings** -> **Control Panel**.
2. Double-click **Add/Remove Programs**.
3. Select **SoftArtisans WordWriter**.
4. Click the **Change/Remove** button to remove the old copy of WordWriter.

Manual Removal

To manually remove WordWriter:

1. **If you are upgrading from V1, remove WordWriter.dll and collections.dll from the Global Assembly Cache:**

   > Installation in the GAC is optional. If WordWriter.dll is not in the GAC, go to step 2.

   - Open a DOS prompt window.
   - Go to the folder OfficeWriter\WordWriter.
   - Enter gacutil.exe /u WordWriter and click Ok.
   - Enter gacutil.exe /u collections and click Ok.
   - Delete the following line from machine.config:

   ```
   <add assembly="WordWriter.dll, Version=x.x.x.x,
    Culture=neutral, PublicKeyToken=f593502af6ee46ae"/>
   ```

   If you are upgrading from V2, remove the SAWW2NET.dll from the Global Assembly Cache:

   > Installation in the GAC is optional. If SAWW2NET.dll is not in the GAC, go to step 2.

   - Open a DOS prompt window.
   - Go to the folder OfficeWriter\WordWriter.
   - Enter gacutil.exe /u SAWW2NET and click Ok.
   - Delete the following line from machine.config:

   ```
   <add assembly="SAWW2NET, Version=x.x.x.x,
    Culture=neutral, PublicKeyToken=f593502af6ee46ae"/>
   ```

   If you are upgrading from V3, remove the SAWW3NET.dll from the Global Assembly Cache:

   > Installation in the GAC is optional. If SAWW3NET.dll is not in the GAC, go to step 2.

   - Open a DOS prompt window.
   - Go to the folder OfficeWriter\WordWriter.
   - Enter gacutil.exe /u SAWW3NET and click Ok.
   - Delete the following line from machine.config:

   ```
   <add assembly="SAWW3NET, Version=x.x.x.x,
    Culture=neutral, PublicKeyToken=f593502af6ee46ae"/>
   ```

   If you are removing V4, remove the SoftArtisans.OfficeWriter.WordWriter.dll from the Global Assembly Cache:

   > Installation in the GAC is optional. If SoftArtisans.OfficeWriter.WordWriter.dll is not in the GAC, go to step 2.

   - Open Windows Explorer.
   - Navigate to the folder C:\WINDOWS\Assembly.
   - Delete SoftArtisans.OfficeWriter.WordWriter and click Ok.
   - Delete the following line from machine.config:

   ```
   <add assembly="SoftArtisans.OfficeWriter.WordWriter, Version=x.x.x.x,
    Culture=neutral, PublicKeyToken=f593502af6ee46ae"/>
   ```

2. **Remove WordWriter's license key:**

   - Double-click **LicenseManager.exe**.

- Select WordWriter's license key.
- Click **Remove Key**

3. **If you are upgrading from V1, unregister WordWriterCOM.dll:**

   - Open a DOS prompt window.
   - Go to the folder OfficeWriter\WordWriter.
   - Enter regsvr32 WordWriterCOM.dll /u and click Ok.
   - To reset IIS, enter iisreset.

   If you are upgrading from V2, unregister SAWW2CCW.dll:

   - Open a DOS prompt window.
   - Go to the folder OfficeWriter\WordWriter.
   - Enter regsvr32 SAWW2CCW.dll /u and click Ok.
   - To reset IIS, enter iisreset.

   If you are upgrading from V3, unregister SAWW3CCW.dll:

   - Open a DOS prompt window.
   - Go to the folder OfficeWriter\WordWriter.
   - Enter regsvr32 SAWW3CCW.dll /u and click Ok.
   - To reset IIS, enter iisreset.

   If you are removing V4, unregister WordTemplateCCW.dll:

   - Open a DOS prompt window.
   - Go to the folder OfficeWriter\WordWriter.
   - Enter regsvr32 WordTemplateCCW.dll /u and click Ok.
   - To reset IIS, enter iisreset.

4. **If you are upgrading from V1, delete these files:**

   - WordWriter.dll
   - collections.dll
   - WordWriterCOM.dll

   > You may have copies of WordWriter.dll and collections.dll in several bin directories.
   > Delete all copies.

   If you are upgrading from V2, delete these files:

   - SAWW2NET.dll
   - SAWW2CCW.dll

   If you are removing V3, delete these files:

   - SAWW3NET.dll
   - SAWW3CCW.dll

   If you are removing V4, delete these files:

   - SoftArtisans.OfficeWriter.WordWriter.dll
   - WordTemplateCCW.dll

   > You may have copies of SAWW2NET.dll, SAWW3NET.dll, or SoftArtisans.OfficeWriter.WordWriter.dll in several bin directories.
   > Delete all copies.

Security Considerations

WordWriter executes in one of the following user contexts:

- The NETWORK SERVICE account (IIS6)
- The ASPNET account (IIS5)
- The authenticated user account
- The impersonated user account (when impersonation is turned on)

When a WordWriter application opens an existing document the application's security context (NETWORK SERVICE, ASPNET, authenticated user, or impersonated user) must have **Read** access to the document.

If you want to save a generated Word file on the server (rather than stream it to a browser), the NETWORK SERVICE account, the ASPNET account, authenticated user, or impersonated user must have **Write** access to the directory in which the file will be saved.

For more information on ASP.NET security, see Authentication in ASP.NET: .NET Security Guidance.

HTMLToWord Object

- Using HTMLToWord
- HTML tags supported by HTMLToWord
- HTMLToWord Programmer's Reference
  - HTMLToWord
    - HTMLToWord.DebugLogLevel
    - HTMLToWord.FormatDelegate
    - HTMLToWord.FormatElementDelegate
    - HTMLToWord.GetImageDelegate
    - HTMLToWord.HTMLInsertProperties
    - HTMLToWord.HTMLInsertProperties.AcceptUnknownTags
    - HTMLToWord.HTMLInsertProperties.DefaultFont
    - HTMLToWord.HTMLInsertProperties.IncludeContentsOfUnknownTags
    - HTMLToWord.HTMLTagAction
    - HTMLToWord.ImageDelegate
    - HTMLToWord.InsertDelegate
    - HTMLToWord.InsertElementDelegate
    - HTMLToWord.InsertHTML
      - HTMLToWord.InsertHTML(String, Document, Element)
      - HTMLToWord.InsertHTML(String, Document, NamedStyle, Element)
      - HTMLToWord.InsertHTML(String, Document)
    - HTMLToWord.InsertProperties
    - HTMLToWord.LogFileName
    - HTMLToWord.LoggingLevel
- The HTMLToWord Object

Using HTMLToWord

- What Is HTMLToWord?
- Requirements
- Installation
  - Obtaining HTMLToWord
  - Building the DLL from the source
  - Setting up your project to use HTMLToWord
- Inserting HTML into a Word document
- Using Delegate Methods
- Troubleshooting
- Participating in the Open Source Project

What is HTMLToWord?

HTMLToWord is an Open Source Microsoft Windows .NET C# project by SoftArtisans, Inc. that allows users to insert fragments of well-formed HTML (XHTML) into a Word document as formatted text. HTMLToWord uses SoftArtisans' OfficeWriter for Word (WordWriter) WordApplication object.

HTMLToWord parses XHTML fragments and inserts the contents into a Word document at the location you specify. In general, you can use HTMLToWord to insert headings, paragraphs, lists and tables into a new or existing Word document that you open or create with the SoftArtisans WordApplication object. The current version of HTMLToWord is not designed to convert entire HTML pages to Word documents.

HTMLToWord understands a subset of the tags defined in the HTML specification. See the documentation page on supported tags for a complete list of supported tags and their support level. For any HTML tag not listed in the table, or if your XHTML documents have custom tags, HTMLToWord will attempt to insert the text contents of the tag into the Word document. You can override this default behavior by setting the AcceptUnknownTags and IncludeContentsOfUnknownTags properties found in the HTMLInsertProperties property container and by implementing the InsertElementDelegate and FormatElementDelegate delegate methods.

Requirements

- OfficeWriter Enterprise Edition

- WordWriter dll (SAWW3NET.DLL or SoftArtisans.OfficeWriter.WordWriter.dll) version 3.8.1.665 or higher

> If you download the binary version of HTML-to-Word, your WordWriter dll must be the current version for which HTML-to-Word is built. Please see the HTML-to-Word open source project homepage for up-to-the-minute WordWriter dll version requirement info.

- OfficeWriter server requirements as described on the SoftArtisans System Requirements page.

Installation

Obtaining HTMLToWord

If you have not already done so, download either the binary version or the source version of the HTMLToWord project from SourceForge.net. Unzip the files into a directory of your choosing.

Building the DLL from the source

If you downloaded the binary version of the project, you can skip this step and move on to Setting up your project to use HTMLToWord.

Create a new Visual C# Class Library project in Visual Studio and include the 3 source files from the source distribution:

- HTMLToWord.cs - The main source file
- AttributeSetter.cs - Helper classes for setting attributes
- XHTMLCleanUp.cs - Helper class for modifying the user-supplied well-formed HTML string to try to make it XHTML-conforming.

In this new project, add a reference to System.XML and to the WordWriter DLL (SAWW3NET.DLL or SoftArtisans.OfficeWriter.WordWriter.dll). Make sure the DLL you have is version 3.8.1 build 665 or later. (If you have an older version of the WordWriter DLL, contact your SoftArtisans sales representative or send an e-mail to sales@softartisans.com.) Press "Ctrl-Shift-B" to build the project. If the build is successful, you will now have a file called HTMLToWord.DLL in your project's Bin\Release directory.

Setting up your project to use HTMLToWord

The HTMLToWord object is defined in the HTMLToWord.DLL file. HTMLToWord makes use of the WordWriter WordApplication object. WordWriter Enterprise Edition must be installed on your system. You must add a reference both to the WordWriter DLL (SAWW3NET.dll or SoftArtisans.OfficeWriter.WordWriter.dll) and to HTMLToWord.DLL in your project. You must have SAWW3NET.DLL or SoftArtisans.OfficeWriter.WordWriter.dll version 3.8.1 build 665 or higher on your system. If you have an older version of the WordWriter DLL, contact your SoftArtisans sales representative or send an e-mail to sales@softartisans.com.

To use the HTMLToWord DLL:

- Make sure you have a reference in your project to the WordWriter DLL (SAWW3NET.DLL or SoftArtisans.OfficeWriter.WordWriter.dll).
- Create a reference in your project to the HTMLToWord DLL (HTMLToWord.DLL).

Inserting HTML into a Word document

Before you begin, you should make sure to import the HTMLToWord namespace in your project.

C#

```
using SoftArtisans.HTMLToWord
```

VB.NET

```
Imports SoftArtisans.HTMLToWord
```

To insert HTML into a WordWriter Document as formatted text, you must first open or create a Document using the SoftArtisans WordApplication object. You will then need to create an instance of the HTMLToWord object. Lastly, you will need some well-formed HTML (XHTML) text to insert into your document. The XHTML can be stored in a file or in a database or can be generated programatically.

In the examples below, we will simply create an XHTML string directly in the code. This text will be inserted at the end of our document.

You must make sure that your HTML is well-formed. That is, every open tag must have a corresponding close tag or be self-closing (e.g.
).

If your HTML is older and does not conform to XHTML standards, you can use any number of utilities, such as the Open Source utility HTMLTidy, to clean-up your HTML before passing it to HTMLToWord's InsertHTML method.

C#

```
//--- Create a new WordWriter document.
WordApplication wApp = new WordApplication();
Document doc = wApp.Create();

//--- Create the HTMLToWord object.
HTMLToWord h2w = new HTMLToWord();

//--- The XHTML to insert into the WordWriter document.
string xhtml = "<h1>Greetings</h1><p><b>Hello</b>, <i>world</i>!</p>";

//--- Insert the XHTML at the end of the WordWriter document.
Element lastInsertedElement = h2w.InsertHTML(xhtml, doc, null, doc);
```

VB.NET

```
'--- Create a new WordWriter document.
Dim wApp As New WordApplication()
Dim doc As Document = wApp.Create()

'--- Create the HTMLToWord object.
Dim h2w As HTMLToWord = New HTMLToWord()

'--- The XHTML to insert into the WordWriter document.
Dim xhtml As String = "<h1>Greetings</h1><p><b>Hello</b>, <i>world</i>!</p>"

'--- Insert the XHTML at the end of the WordWriter document.
Dim lastInsertedElement As Element = h2w.InsertHTML(xhtml, doc, null, doc)
```

Using Delegate Methods

Delegate methods enable you to write custom code that is called by HTMLToWord while processing an HTML string. If you specify them, delegate methods can be called:

1. When a tag in the XHTML string is encountered. This allows you to optionally process the tag yourself before handing control back to HTMLToWord. You can also tell HTMLToWord how to handle each tag: process the tag and all of its contents and sub-tags; or skip over this tag and all of the sub-tags.
   See documentation for InsertDelegate delegate.
2. After an Element has been inserted into the document, HTMLToWord gives you the opportunity to do additional processing on the new element, such as adding formatting.
   See documentation for FormatDelegate.
3. Whenever an HTML img tag is encountered. HTMLToWord itself does not know how to find image file that is specified with an HTML img tag. Using a delegate method, HTMLToWord looks to you to find the image file's contents and pass it back as a System.IO.Stream.
   See documentation for ImageDelegate delegate.

HTMLToWord includes three properties that allow you to specify your own methods for handling HTML tags.

- InsertDelegate - method called by HTMLToWord each time it encounters a tag in the HTML string.
- FormatDelegate - method called by HTMLToWord after it has inserted an element into the Word document.
- ImageDelegate - method called by HTMLToWord each time it encounters an img tag.

Using delegate methods, you can add handling for any tag in the XHTML string, even tags that are not part of HTML.

If you implement a InsertElementDelegate or FormatElementDelegate delegate method, you must import the System.Xml namespace into your project. HTMLToWord turns your XHTML string into an XML DOM tree, and hands you an XmlNode object representing the HTML tag being processed.

The example below shows how to create a delegate method that will be called each time an HTML is encountered by HTMLToWord and assign it to the InsertDelegate property. The delegate method does 2 things. First, it handles a custom para tag that is in our HTML by inserting a paragraph into the Word document. Second, it tells HTMLToWord to ignore anything inside a custom private tag so that sensitive information is not written into the document.

C#

```
//--- This method will be used as a delegate by HTMLToWord and
//--- is an example of type HTMLToWord.InsertElementDelegate.

//--- We have defined 2 custom XHTML tags for our documents: para and private.
//--- If we encounter a <para> tag, we insert a paragraph into the Word document and tell HTMLToWord
//--- to continue processing.  If we encounter a <private> tag, we write a short message into
//--- the Word document and tell HTMLToWord to skip this tag altogether.
HTMLToWord.HTMLTagAction MyInserter(Document doc, Element insertAfterElement, XmlNode node)
{
     //--- Get the name of the HTML tag and convert it to lower case.
     string nodeName = node.Name.ToLower();

     if (nodeName.equals("para"))
     {
          //--- There is a special <para> tag in our markup, but we should just treat it
          //--- like a standard HTML <p> tag.  Insert a paragraph into the document and
          //--- let HTML continue processing.
          insertAfterElement.InsertParagraphAfter(null);
     }
     else if (nodeName.equals("private"))
     {
          //--- The "private" tag is used to indicate information that should not be sent
          //--- to outside clients; therefore, we tell HTMLToWord to skip this tag and its
          //--- contents.
          return HTMLToWord.HTMLTagAction.Skip;
     }

     //--- We have handled all of the special cases.  Let HTMLToWord know that
     //--- it should process this tag.
     return HTMLToWord.HTMLTagAction.Process;
}
```

VB.NET

```
'--- This method will be used as a delegate by HTMLToWord and
'--- is an example of type HTMLToWord.InsertElementDelegate.

'--- We have defined 2 custom XHTML tags for our documents: para and private.
'--- If we encounter a <para> tag, we insert a paragraph into the Word document and tell HTMLToWord
'--- to continue processing.  If we encounter a <private> tag, we write a short message into
'--- the Word document and tell HTMLToWord to skip this tag altogether.
Function MyInserter(ByVal doc As Document, ByVal insertAfterElement As Element, ByVal xmlNode As
XmlNode)
     As HTMLToWord.HTMLTagAction

     '--- Get the name of the HTML tag and convert it to lower case.
     Dim nodeName As String = node.Name.ToLower()

     If (nodeName.Equals("para")) Then
          '--- There is a special <para> tag in our markup, but we should just treat it
          '--- like a standard HTML <p> tag.  Insert a paragraph into the document and
          '--- let HTML continue processing.
          insertAfterElement.InsertParagraphAfter(Nothing)
     Else If (nodeName.Equals("private"))
          '--- The "private" tag is used to indicate information that should not be sent
          '--- to outside clients; therefore, we tell HTMLToWord to skip this tag and its
          '--- contents.
          Return HTMLToWord.HTMLTagAction.Skip
     End If

     '--- We have handled all of the special cases.  Let HTMLToWord know that
     '--- it should process this tag.
     Return HTMLToWord.HTMLTagAction.Process
End Function
```

You must tell HTMLToWord about your delegate method so that HTMLToWord can call your custom code. To do that, assign your delegate method to the appropriate HTMLToWord property.

C#

```
//--- Create an HTMLToWord instance.
    HTMLToWord h2w = new HTMLToWord();

    //--- Using the code from the example above, tell HTMLToWord that it should
    //--- call our "MyInserter" delegate method whenever it encounters a tag in
    //--- the XHTML string.
    h2w.InsertDelegate = new HTMLToWord.InsertElementDelegate(MyInserter);
```

VB.NET

```
'--- Create an HTMLToWord instance.
    Dim h2w As New HTMLToWord()

    '--- Using the code from the example above, tell HTMLToWord that it should
    '--- call our "MyInserter" delegate method whenever it encounters a tag in
    '--- the XHTML string.
    h2w.InsertDelegate = New HTMLToWord.InsertElementDelegate(AddressOf MyInserter)
```

Note that HTMLToWord does not allow you to chain delegate methods.

Troubleshooting

This section contains information to help you if you are having trouble using the HTMLToWord.DLL in your project.

Error: The located assembly's manifest definition with name 'SAWW3NET' does not match the assembly reference.

Exact error message: java.lang.ClassNotFoundException: The located assembly's manifest definition with name 'SAWW3NET' does not match the assembly reference.

This error usually indicates that the version of the WordWriter DLL that you have on your system does not match the version that was used to create the HTMLToWord.DLL file.

If you downloaded the binary version of the project from the SourceForge web site, check the ReadMe.txt file in the directory where you unzipped the distribution. It contains the version number of the WordWriter DLL (SAWW3NET.DLL) that was used to create the HTMLToWord.DLL file. If the version of the WordWriter DLL on your system is older than the version documented in the ReadMe.txt file, you should contact your SoftArtisans sales representative or send an e-mail to sales@softartisans.com to get an newer version of the WordWriter DLL.

If you downloaded the source version of the project from the SourceForge web site, make sure that the version of the WordWriter DLL you used to build the HTMLToWord.DLL file is the same version and build number as the version of the WordWriter DLL file (SAWW3NET.DLL) you are referencing in your project.

Participating in the Open Source Project

SoftArtisans has released the source code for HTMLToWord as an Open Source project. If you would like to download the lastest version of HTMLToWord in either binary format (the DLL plus user documentation for the API) or source code (the Visual C# source files plus user documentation for the API), feel free to visit our project page on SourceForge.net at http://sourceforge.net/projects/htmltoword. Note that the binary version will work on 32-bit platforms with .NET 1.1 or later. If you would like a build for 64-bit platforms, please download the source distribution and rebuild the DLL on your system.

HTML tags supported by HTMLToWord

The following table lists the HTML tags that HTMLToWord explicitly supports.

The contents of HTML tags not listed in the table below (or custom XHTML tags in your document) will be inserted into the document as text. You can control this default behavior by setting the AcceptUnknownTags and IncludeContentsOfUnknownTags properties found in the InsertHTMLProperties property container and by implementing the InsertDelegate and FormatDelegate delegate methods.

HTML Tag	Description	Support Level

a	Inserts a hyperlink. **Supported:** HTTP-protocol hrefs **Unsupported:** https, mailto or ftp protocols in the href attribute and the name attribute.	Partial
b	Renders the text as bold.	Full
blockquote	Offsets text and narrows the left and right margins.	Full
br	Insert a line break.	Full
code	Renders the text in-line with a mono-spaced font.	Full
del	Renders text as strike-through. **Supported:** In-line elements. **Unsupported:** Block-level elements.	Partial
dir	Inserts an unordered list into the document. (Note, this tag is deprecated as of HTML 4.01.)	Full
div	Mimics HTML by separating content from surrounding text. **Unsupported:** Does not support any attributes.	Partial
em	Renders the text as italics.	Full
font	Changes the font face, font size and/or font color. Size can be either absolute or relative. If given a comma-separated list of faces, uses the first face only. Color can be either a name or an HTML code, such as #FFFFFF for white.	Full
h1 . . . h6	Applies styles corresponding to Microsoft Word styles H1 through H6. **Supported:** Heading levels 1-6. **Unsupported:** Align attribute.	Partial
i	Italicize text.	Full
img	Inserts an image in-line with the text. Relies on the GetImageDelegate method to retrieve the actual data bits of the image. **Supported:** In-line images. Height and width attributes as absolute pixel sizes. **Unsupported:** Floating or absolutely positioned images. Height and width specified as percentage, usemap attribute.	Partial
ins	Renders text as underlined. **Supported:** In-line elements. **Unsupported:** Block-level elements.	Partial
li	Creates a new item in the enclosing list. **Unsupported:** Attributes.	Partial
menu	Inserts an unordered list into the document. (Note, this tag is deprecated as of HTML 4.01.)	Full
ol	Ordered (numbered) list. **Unsupported:** Attributes.	Partial
p	Paragraph. The align attribute is supported.	Full
pre	Renders text in a mono-spaced font. Offsets block of text from surrounding text with an extra blank line before and after. **Unsupported:** The width attribute is ignored.	Partial
s	Renders text as strikethrough.	Full
span	Used to specify characteristics for a section of text with a block.	Full
strike	Renders text as strikethrough.	Full

strong	Renders text as bold.	Full
style	HTMLToWord does not parse the style tag; however, users can parse a style using a delegate method. **Supported:** User-supplied delegate method to handle styles. **Unsupported:** Automatic parsing of styles.	Partial
sub	Renders text as subscript	Full
sup	Renders text as superscript	Full
table	Inserts a table into the document. Nested tables are supported. **Supported:** bgcolor, border, cellpadding, cellspacing, frame, rules, and width attributes. Cellpadding and cellspacing attributes with absolute size values. **Unsupported:** Cellpadding and cellspacing attributes with percentage values.	Partial
tbody	Body of a table. **Supported:** tbody tags within the XHTML to denote the start of the body of the table. **Unsupported:** Cell alignment attributes: align, char, charoff and valign.	Partial
td	Specifies the contents of a table cell. **Supported:** The colspan, rowspan, align, valign, and bgcolor attributes. **Unsupported:** The abbr, axis, headers, scope, nowrap, width, height, char and charoff attributes.	Partial
tr	Table row.	Full
tt	Renders text in a serifed, mono-spaced font (Courier New).	Full
u	Renders text as underlined.	Full
ul	Unordered (bulleted) list. **Unsupported:** Attributes.	Partial

HTMLToWord Programmer's Reference

- HTMLToWord — Use the HTMLToWord object to insert well-formed HTML (XHTML) text into a WordWriter Document object. You can specify a style to use when inserting text or use the document's default style. You can also specify whether to insert text after a specific element or at the end of a document.

HTMLToWord

Description

Use the HTMLToWord object to insert well-formed HTML (XHTML) text into a WordWriter Document object. You can specify a style to use when inserting text or use the document's default style. You can also specify whether to insert text after a specific element or at the end of a document.

Signature: C#

```
public class HTMLToWord
```

Signature: VB.NET

```
Public Class HTMLToWord
```

Remarks

The contents of HTML tags not listed on the HTMLToWord Supported Tags may be inserted into the document as plain text. The behavior is dependent on the settings of the AcceptUnknownTags and IncludeContentsOfUnknownTags properties found in the HTMLInsertProperties

property container.

For an overview of how to use the HTMLToWord object, see Using HTMLToWord.

Examples

C#

```
//--- Well-formed HTML fragment to be inserted into the document.
string htmlFragment = @"<{span}{/span}p><{span}{/span}b>Hello,<{span}{/span}/b><{span}{/span}i>
World<{span}{/span}/i>!<{span}{/span}/p>";


//--- Open an existing Word document with the SoftArtisans WordApplication object.
WordApplication wApp = new WordApplication();
Document doc = wApp.Open("myDoument.doc");

//--- Create the HTMLToWord object.
HTMLToWord h2w = new HTMLToWord();

//--- Insert the HTML fragment at the end of the document.
//--- Use the default style.
h2w.InsertHTML(htmlFragment, doc, null, doc);
```

vb.net

```
'--- Well-formed HTML fragment to be inserted into the document.
Dim htmlFragment As String = "<{span}{/span}p><{span}{/span}b>Hello,<{span}{/span}/b><{span}{/span}i>
World<{span}{/span}/i>!<{span}{/span}/p>"

'--- Open an existing Word document with the SoftArtisans WordApplication object.
Dim wApp As New WordApplication()
Dim doc As Document = wApp.Open("myDoument.doc")

'--- Create the HTMLToWord object.
Dim h2w As New HTMLToWord()

'--- Insert the HTML fragment at the end of the document.
'--- Use the default style.
h2w.InsertHTML(htmlFragment, doc, Nothing, doc)
```

Fields

Name	Description
FormatElementDelegate	Type for user-defined delegate method called by HTMLToWord after inserting an element.
GetImageDelegate	Type for user-defined delegate method called by HTMLToWord when an HTML `img` tag is encountered.
InsertElementDelegate	Type for user-defined delegate method called by HTMLToWord before inserting an element.

Properties

Name	Description
DebugLogLevel	Sets the amount of information written to the debug log file. To create a log file, you must also specify the full path of the file in the HTMLToWord.LogFileName property. If no errors occur then HTMLToWord does not generate a log file.
FormatDelegate	User-assigned delegate method that is called after HTMLToWord inserts an Element into the Word document. The user may implement code to perform additional formatting or post-processing on the Element inserted into the document. If this delegate is not set, HTMLToWord simply continues processing.
ImageDelegate	User-assigned delegate method that allows users to provide an implementation for retrieving the contents of an image and passing it back to HTMLToWord for insertion into a document. If not set, then HTMLToWord will not insert images into a document.

InsertDelegate	A user-assigned delegate method that HTMLToWord calls before it processes an XHTML tag. The user may implement code to perform any tasks they desire before passing control back to HTMLToWord. The return value from the delegate method specifies how HTMLToWord should behave. If this property is not set, HTMLToWord will attempt to insert the contents of the XHTML tag.
InsertProperties	Contains settings that control how unsupported HTML tags are treated and provides an override to the default font used when inserting text into the document.
LogFileName	The full path to the HTMLToWord debug log file. To create a log file, you must set the DebugLogLevel parameter to a value other than None.

Methods

Name	Description
InsertHTML(String, Document, NamedStyle, Element)	Processes the XHTML string and inserts it into the specified Word document as formatted text after the given WordWriter Element using the given NamedStyle. Returns the last Element inserted into the document.
InsertHTML(String, Document, Element)	Processes the XHTML string and inserts it into the specified Word document as formatted text after the given WordWriter Element using the document's Normal style as the default font. Returns the last Element inserted into the document.
InsertHTML(String, Document)	Processes the XHTML string and inserts it at the end of the specified Word document as formatted text using the document's Normal style as the default font. Returns the last Element inserted into the document.

Nested Classes

Name	Description
HTMLInsertProperties	The HTMLInsertProperties object is a container for properties that control the behavior of HTMLToWord. You can control how HTMLToWord handles HTML tags that it does not explicitly support, as well as set the default font used when inserting text into the Word document.
HTMLTagAction	Return value from the user-supplied InsertElementDelegate delegate method. Controls how HTMLToWord handles an XHTML tag.
LoggingLevel	A LoggingLevel value specified the amount of information written to the log file. To create a log file, you must also specify the full path for the file using the LogFileName property.

HTMLToWord.DebugLogLevel

Description

Sets the amount of information written to the debug log file. To create a log file, you must also specify the full path of the file in the HTMLToWord.LogFileName property. If no errors occur then HTMLToWord does not generate a log file.

Signature: C#

```
public LoggingLevel DebugLogLevel{ get; set; }
```

Signature: VB.NET

```
Public Property DebugLogLevel() As LoggingLevel
```

Remarks

For a list of valid property values, see HTMLToWord.LoggingLevel.

Examples

C#

```
//--- Get the current logging level.
HTMLToWord.LoggingLevel currentLogLevel = h2w.DebugLogLevel;

//--- Set the log level of the h2w object equal to Debug
h2w.LoggingLevel = HTMLToWord.LoggingLevel.Debug;
h2w.LogFileName = @"c:\HTMLToWord\Logfile.txt";
```

vb.net

```
'--- Get the current logging level.
Dim currentLogLevel As HTMLToWord.LoggingLevel = h2w.DebugLogLevel

'--- Set the log level of the oHTMLToWord object equal to Trace
Dim h2w As New HTMLToWord()
h2w.DebugLogLevel = HTMLToWord.LoggingLevel.Trace
h2w.LogFileName = "c:\HTMLToWord\Logfile.txt"
```

HTMLToWord.FormatDelegate

Description

User-assigned delegate method that is called after HTMLToWord inserts an Element into the Word document. The user may implement code to perform additional formatting or post-processing on the Element inserted into the document. If this delegate is not set, HTMLToWord simply continues processing.

Signature: C#

```
public FormatElementDelegate FormatDelegate{ get; set; }
```

Signature: VB.NET

```
Public Property FormatDelegate() As FormatElementDelegate
```

Examples

C#

```
void MyFormatter(Element insertedElement, XmlNode node)
 {
      string nodeName = node.Name.ToLower();

      if (nodeName.Equals("ul"))
      {
           //--- First, cast the WordWriter Element object (passed in as a parameter to this method)
           //--- to a WordWriter List object.
           List theList = (List)insertedElement;

           //--- We want bulleted lists to have 1 line of spacing above and below to offset them
           //--- from the text.  This mimics the behavior of most HTML renderers.

           //--- Get the font size used in the list.  We'll use that value to determine
           //--- the appropriate line spacing gap.
           double fontSize = theList.GetEntry(0).Style.Font.FontSize;
           theList.GetEntry(0).Formatting.SpaceBefore = TwipsConverter.FromPoints(fontSize);
           theList.GetEntry(theList.NumEntries - 1).Formatting.SpaceAfter =
                TwipsConverter.FromPoints(fontSize);
      }
 }


 //--- Create an HTMLToWord instance.
 HTMLToWord h2w = new HTMLToWord();

 //--- Tell HTMLToWord that it should call our "MyFormatter" delegate method
 //--- whenever an Element is inserted into the Word document.
 h2w.FormatDelegate = new HTMLToWord.FormatElementDelegate(MyFormatter);
```

vb.net

```
Sub MyFormatter(Element insertedElement, XmlNode node)
      Dim nodeName As String = node.Name.ToLower()

      If nodeName.Equals("ul") Then
           '--- First, cast the WordWriter Element object (passed in as a parameter to this method)
           '--- to a WordWriter List object.
           Dim theList As List = insertedElement

           '--- We want bulleted lists to have 1 line of spacing above and below to offset them
           '--- from the text.  This mimics the behavior of most HTML renderers.

           '--- Get the font size used in the list.  We'll use that value to determine
           '--- the appropriate line spacing gap.
           Dim fontSize As Double = theList.GetEntry(0).Style.Font.FontSize
           theList.GetEntry(0).Formatting.SpaceBefore = TwipsConverter.FromPoints(fontSize)
           theList.GetEntry(theList.NumEntries - 1).Formatting.SpaceAfter = _
                TwipsConverter.FromPoints(fontSize)
      End If
 End Sub



 '--- Create an HTMLToWord instance.
 Dim h2w As New HTMLToWord()

 '--- Tell HTMLToWord that it should call our "MyFormatter" delegate method
 '--- whenever an Element is inserted into the Word document.
 h2w.FormatDelegate = New HTMLToWord.FormatElementDelegate(AddressOf MyFormatter)
```

HTMLToWord.FormatElementDelegate

Description

Type for user-defined delegate method called by HTMLToWord after inserting an element.

Signature: C#

```
public delegate void FormatElementDelegate(SoftArtisans.OfficeWriter.WordWriter.Element
newlyInsertedElement, System.Xml.XmlNode node);
```

Signature: VB.NET

```
Public Delegate Sub FormatElementDelegate(ByVal newlyInsertedElement As
SoftArtisans.OfficeWriter.WordWriter.Element, ByVal node As System.Xml.XmlNode)
```

HTMLToWord.GetImageDelegate

Description

Type for user-defined delegate method called by HTMLToWord when an HTML `img` tag is encountered.

Signature: C#

```
public delegate System.IO.Stream GetImageDelegate(System.String URI);
```

Signature: VB.NET

```
Public Delegate Function GetImageDelegate(ByVal URI As String) As System.IO.Stream
```

HTMLToWord.HTMLInsertProperties

Description

The HTMLInsertProperties object is a container for properties that control the behavior of HTMLToWord. You can control how HTMLToWord handles HTML tags that it does not explicitly support, as well as set the default font used when inserting text into the Word document.

Signature: C#

```
public class HTMLInsertProperties
```

Signature: VB.NET

```
Public Class HTMLInsertProperties
```

Remarks

You can not create an `HTMLInsertProperties` object. You must obtain the existing object from the HTMLToWord object by getting HTMLToWord.InsertProperties.

Properties

Name	Description
AcceptUnknownTags	Controls how unknown tags are handled. If set to false, HTMLToWord will throw an HTMLToWordException if it encounters a tag that it does not explicitly know how to handle. If set to true, HTMLToWord will attempt to process the tag. See the IncludeContentsOfUnknownTags for further user-controlable behavior.
DefaultFont	Specifies the font to use when inserting text. This property is temporarily overridden by the <font> tag in the XHTML. If this property is not set, the document's default font is used.
IncludeContentsOfUnknownTags	Controls how the contents of unsupported tags are handled. If set to true, the contents of unsupported tags are inserted into the document as text. If set to false, the contents of unsupported tags are ignored.

HTMLToWord.HTMLInsertProperties.AcceptUnknownTags

Description

Controls how unknown tags are handled. If set to false, HTMLToWord will throw an HTMLToWordException if it encounters a tag that it does not explicitly know how to handle. If set to true, HTMLToWord will attempt to process the tag. See the IncludeContentsOfUnknownTags for further user-controlable behavior.

Signature: C#

```
public boolean AcceptUnknownTags{ get; set; }
```

Signature: VB.NET

```
Public Property AcceptUnknownTags() As Boolean
```

Examples

C#

```
//--- Create the HTMLToWord object
  HTMLToWord h2w = new HTMLToWord();

  //--- Set HTMLToWord to throw an exception if there are any tags
  //--- in the XHTML fragment that are not explicitly handled.
  h2w.InsertProperties.AcceptUnknownTags = false;
```

vb.net

```
'--- Create the HTMLToWord object
  Dim h2w As New HTMLToWord()

  '--- Set HTMLToWord to throw an exception if there are any tags
  '--- in the XHTML fragment that are not explicitly handled.
  h2w.InsertProperties.AcceptUnknownTags = False
```

HTMLToWord.HTMLInsertProperties.DefaultFont

Description

Specifies the font to use when inserting text. This property is temporarily overridden by the <font> tag in the XHTML. If this property is not set, the document's default font is used.

Signature: C#

```
public SoftArtisans.OfficeWriter.WordWriter.Font DefaultFont{ get; set; }
```

Signature: VB.NET

```
Public Property DefaultFont() As SoftArtisans.OfficeWriter.WordWriter.Font
```

Remarks

The default value is **null**. The font in the document's default style will be used. (This is generally Times New Roman 12 pt.)

Examples

C#

```
//--- Create a WordApplication and open a Document
  WordApplication wordApp = new WordApplication();
  Document doc = wordApp.Create();

  //--- Create the HTMLToWord object
  HTMLToWord h2w = new HTMLToWord();

  //--- Create the font to use when inserting HTML
  //--- Start by getting the "Normal" font from the Document
  Font font = doc.Styles[NamedStyle.BuiltIn.Normal].Font;

  //--- Change the font to 10 pt. Arial with red text
  font.FontName = "Arial";
  font.FontSize = 10;
  font.TextColor = System.Drawing.Color.Red;

  //--- Set this as the default font on the HTMLToWord object
  h2w.InsertProperties.DefaultFont = font;

  //--- Insert the htmlFragment at the end of the document.
  h2w.InsertHTML(htmlFragment, doc, Nothing, doc);
```

vb.net

```
'--- Create a WordApplication and open a Document
  Dim wordApp As New WordApplication()
  Dim doc As Document = wordApp.Create()

  '--- Create the HTMLToWord object
  Dim h2w As New HTMLToWord()

  '--- Create the font to use when inserting HTML
  '--- Start by getting the "Normal" font from the Document
  Dim font as Font = doc.Styles(NamedStyle.BuiltIn.Normal).Font

  '-- Change the font to 10 pt. Arial with red text
  font.FontName = "Arial"
  font.FontSize = 10
  font.TextColor = System.Drawing.Color.Red

  '--- Set this as the default font on the HTMLToWord object
  h2w.InsertProperties.DefaultFont = font

  '--- Insert the htmlFragment at the end of the document.
  h2w.InsertHTML(htmlFragment, doc, Nothing, doc)
```

HTMLToWord.HTMLInsertProperties.IncludeContentsOfUnknownTags

Description

Controls how the contents of unsupported tags are handled. If set to true, the contents of unsupported tags are inserted into the document as text. If set to false, the contents of unsupported tags are ignored.

Signature: C#

```
public boolean IncludeContentsOfUnknownTags{ get; set; }
```

Signature: VB.NET

```
Public Property IncludeContentsOfUnknownTags() As Boolean
```

Remarks

The default value is **true**.

Examples

C#

```
//--- Create the HTMLToWord object
  HTMLToWord h2w = new HTMLToWord();

  //--- Set HTMLToWord to ignore the contents of any XHTML tags
  //--- that are not explicitly handled.
  h2w.InsertProperties.IncludeContentsOfUnknownTags = false;
```

vb.net

```
'--- Create the HTMLToWord object
  Dim h2w As New HTMLToWord()

  '--- Set HTMLToWord to ignore the contents of any XHTML tags
  '--- that are not explicitly handled.
  h2w.InsertProperties.IncludeContentsOfUnknownTags = False
```

HTMLToWord.HTMLTagAction

Description

Return value from the user-supplied InsertElementDelegate delegate method. Controls how HTMLToWord handles an XHTML tag.

Signature: C#

```
public sealed enum HTMLTagAction
```

Signature: VB.NET

```
Public NotInheritable Enum HTMLTagAction
 End Enum
```

Values

Name	Description
Process	Instructs HTMLToWord to insert the contents of this tag and all of its sub-tags.
Skip	Instructs HTMLToWord to ignore this tag and all of its sub-tags and move to the next tag at the same level (sibling).

HTMLToWord.ImageDelegate

Description

User-assigned delegate method that allows users to provide an implementation for retrieving the contents of an image and passing it back to HTMLToWord for insertion into a document. If not set, then HTMLToWord will not insert images into a document.

Signature: C#

```
public GetImageDelegate ImageDelegate{ get; set; }
```

Signature: VB.NET

```
Public Property ImageDelegate() As GetImageDelegate
```

Examples

C#

```
System.IO.Stream MyImageGetter(string URI)
 {
      //--- Check if there is data in the supplied URI.
      if (URI == null || URI.Length == 0)
      {
           //--- No URI or empty URI.  Nothing for us to do.
           //--- Indicate that there are no image contents.
           return null;
      }

      //--- Create a web request object to fetch the image file.
      WebRequest request;
      try
      {
           request = WebRequest.Create(URI);
           request.Method = "GET";
           request.Timeout = 5000;   // in milliseconds
      }
      catch (Exception e)
      {
           Console.WriteLine("Image (" + URI + ") ignored: " + e.Message);
           return null;
      }

      //--- Get the file and return the stream.
      System.IO.Stream s = null;
      try
      {
           s = request.GetResponse().GetResponseStream();
      }
      catch (WebException e)
      {
           Console.WriteLine("Image (" + URI + ") ignored: " + e.Message);
           return null;
      }
      catch (Exception e)
      {
           Console.WriteLine("Image (" + URI + ") ignored: " + e.Message);
           return null;
      }

      //--- We have successfully retrieved the image contents.
      //--- Pass the stream back to HTMLToWord.
      return s;
 }



 //--- Create an HTMLToWord instance.
 HTMLToWord h2w = new HTMLToWord();

 //--- Tell HTMLToWord that it should call our "MyImageGetter" delegate method
 //-- whenever it encounters an img tag in the XHTML string.
 h2w.ImageDelegate = new HTMLToWord.GetImageDelegate(MyImageGetter);
```

vb.net

```
Function MyImageGetter(string URI) As System.IO.Stream
      '--- Check if there is data in the supplied URI.
      If URI Is Nothing Or URI.Length = 0 Then
           '--- No URI or empty URI.  Nothing for us to do.
           '--- Indicate that there are no image contents.
           return Nothing
      End If

      '--- Create a web request object to fetch the image file.
      Dim request As WebRequest
      Try
           request = WebRequest.Create(URI)
           request.Method = "GET"
           request.Timeout = 5000   ' in milliseconds
      Catch e As Exception
           return Nothing
      End Try

      '--- Get the file and return the stream.
      Dim s As System.IO.Stream = Nothing
      Try
           s = request.GetResponse().GetResponseStream()
      Catch e As WebException
           return Nothing
      Catch e As Exception
           return Nothing
      End Try

      '--- We have successfully retrieved the image contents.
      '--- Pass the stream back to HTMLToWord.
      return s
 End Function



 '--- Create an HTMLToWord instance.
 Dim h2w As New HTMLToWord()

 '--- Tell HTMLToWord that it should call our "MyImageGetter" delegate method
 '--- whenever it encounters an img tag in the XHTML string.
 h2w.ImageDelegate = New HTMLToWord.GetImageDelegate(AddressOf MyImageGetter)
```

HTMLToWord.InsertDelegate

Description

A user-assigned delegate method that HTMLToWord calls before it processes an XHTML tag. The user may implement code to perform any tasks they desire before passing control back to HTMLToWord. The return value from the delegate method specifies how HTMLToWord should behave. If this property is not set, HTMLToWord will attempt to insert the contents of the XHTML tag.

Signature: C#

```
public InsertElementDelegate InsertDelegate{ get; set; }
```

Signature: VB.NET

```
Public Property InsertDelegate() As InsertElementDelegate
```

Examples

C#

```
//--- We have defined two custom HTML tags for our documents: para and private.
 //--- If we encounter a &lt;para> tag, we insert a paragraph into the Word document and tell
HTMLToWord
 //--- to continue processing.  If we encounter a &lt;private> tag, we write a short message into
 //--- the Word document and tell HTMLToWord to skip this tag altogether.
 HTMLToWord.HTMLTagAction MyInserter(Document doc, Element insertAfterElement, XmlNode node)
 {
      //--- Get the name of the HTML tag and convert it to lower case.
      string nodeName = node.Name.ToLower();

      if (nodeName.equals("para"))
      {
           //--- There is a special &lt;para> tag in our markup, but we should just treat it
           //--- like a standard HTML &lt;p> tag.  Insert a paragraph into the document and
           //--- let HTML continue processing.
           insertAfterElement.InsertParagraphAfter(null);
      }
      else if (nodeName.equals("private"))
      {
           //--- The "private" tag is used to indicate information that should not be sent
           //--- to outside clients; therefore, we tell HTMLToWord to skip this tag and its
           //--- contents.
           return HTMLToWord.HTMLTagAction.Skip;
      }

      //--- We have handled all of the special cases.  Let HTMLToWord know that
      //--- it should process this tag.
      return HTMLToWord.HTMLTagAction.Process;
 }


 //--- Create an HTMLToWord instance.
 HTMLToWord h2w = new HTMLToWord();

 //--- Tell HTMLToWord that it should call our "MyInserter" delegate method
 //--- whenever it encounters a tag in the XHTML string.
 h2w.InsertDelegate = new HTMLToWord.InsertElementDelegate(MyInserter);
```

vb.net

```
'--- We have defined two custom XHTML tags for our documents: para and private.
 '--- If we encounter a &lt;para> tag, we insert a paragraph into the Word document and tell
HTMLToWord
 '--- to continue processing.  If we encounter a &lt;private> tag, we write a short message into
 '--- the Word document and tell HTMLToWord to skip this tag altogether.
 Function MyInserter(ByVal doc As Document, ByVal insertAfterElement As Element, ByVal xmlNode As
XmlNode)
   As HTMLToWord.HTMLTagAction
 {
      '--- Get the name of the HTML tag and convert it to lower case.
      Dim nodeName As String = node.Name.ToLower()

      If nodeName.Equals("para") Then
           '--- There is a special &lt;para> tag in our markup, but we should just treat it
           '--- like a standard HTML &lt;p> tag.  Insert a paragraph into the document and
           '--- let HTML continue processing.
           insertAfterElement.InsertParagraphAfter(Nothing)
      Else If nodeName.Equals("private")
           '--- The "private" tag is used to indicate information that should not be sent
           '--- to outside clients; therefore, we tell HTMLToWord to skip this tag and its
           '--- contents.
           Return HTMLToWord.HTMLTagAction.Skip;
      End If

      '--- We have handled all of the special cases.  Let HTMLToWord know that
      '--- it should process this tag.
      Return HTMLToWord.HTMLTagAction.Process
 End Function



 '--- Create an HTMLToWord instance.
 Dim h2w As New HTMLToWord()

 '--- Tell HTMLToWord that it should call our "MyInserter" delegate method
 '--- whenever it encounters a tag in the XHTML string.
 h2w.InsertDelegate = New HTMLToWord.InsertElementDelegate(AddressOf MyInserter)
```

HTMLToWord.InsertElementDelegate

Description

Type for user-defined delegate method called by HTMLToWord before inserting an element.

Signature: C#

```
public delegate HTMLTagAction InsertElementDelegate(SoftArtisans.OfficeWriter.WordWriter.Document doc,
SoftArtisans.OfficeWriter.WordWriter.Element insertAfterElement, System.Xml.XmlNode node);
```

Signature: VB.NET

```
Public Delegate Function InsertElementDelegate(ByVal doc As
SoftArtisans.OfficeWriter.WordWriter.Document, ByVal insertAfterElement As
SoftArtisans.OfficeWriter.WordWriter.Element, ByVal node As System.Xml.XmlNode) As HTMLTagAction
```

HTMLToWord.InsertHTML

Overload	Description
InsertHTML(String, Document)	Processes the XHTML string and inserts it at the end of the specified Word document as formatted text using the document's Normal style as the default font. Returns the last Element inserted into the document.

InsertHTML(String, Document, Element)	Processes the XHTML string and inserts it into the specified Word document as formatted text after the given WordWriter Element using the document's Normal style as the default font. Returns the last Element inserted into the document.
InsertHTML(String, Document, NamedStyle, Element)	Processes the XHTML string and inserts it into the specified Word document as formatted text after the given WordWriter Element using the given NamedStyle. Returns the last Element inserted into the document.

HTMLToWord.InsertHTML(String, Document, Element)

Description

Processes the XHTML string and inserts it into the specified Word document as formatted text after the given WordWriter Element using the document's Normal style as the default font. Returns the last Element inserted into the document.

Signature: C#

```
public SoftArtisans.OfficeWriter.WordWriter.Element InsertHTML(System.String xhtml,
SoftArtisans.OfficeWriter.WordWriter.Document doc, SoftArtisans.OfficeWriter.WordWriter.Element
insertAfterElement)
```

Signature: VB.NET

```
Public Function InsertHTML(ByVal xhtml As String, ByVal doc As
SoftArtisans.OfficeWriter.WordWriter.Document, ByVal insertAfterElement As
SoftArtisans.OfficeWriter.WordWriter.Element) As SoftArtisans.OfficeWriter.WordWriter.Element
```

Parameters

xhtml

A well-formed fragment of an xhtml document.

doc

a WordWriter document object into which content will be inserted.

insertAfterElement

The WordWriter element after which the contents of the xhtml will be inserted.

Returns

The last Element inserted into the document.

Examples

C#

```
//--- Create a new WordWriter document.
 WordApplication wApp = new WordApplication();
 Document doc = wApp.Create();

 //--- Create the HTMLToWord object.
 HTMLToWord h2w = new HTMLToWord();

 //--- The XHTML to insert into the WordWriter document.
 string xhtml = "&lt;h1>Greetings&lt;/h1>&lt;p>&lt;b>Hello&lt;/b>, &lt;i>world&lt;/i>!&lt;/p>";

 //--- Insert the XHTML at the end of the WordWriter document.
 Element lastInsertedElement = h2w.InsertHTML(xhtml, doc, null, doc);
```

vb.net

```
'--- Create a new WordWriter document.
 Dim wApp As New WordApplication()
 Dim doc As Document = wApp.Create()

 '--- Create the HTMLToWord object.
 Dim h2w As New HTMLToWord()

 '--- The XHTML to insert into the WordWriter document.
 Dim xhtml As String = "&lt;h1>Greetings&lt;/h1>&lt;p>&lt;b>Hello&lt;/b>, &lt;i>world&lt;/i>!&lt;/p>"

 '--- Insert the XHTML at the end of the WordWriter document.
 Dim lastInsertedElement As Element = h2w.InsertHTML(xhtml, doc, null, doc)
```

HTMLToWord.InsertHTML(String, Document, NamedStyle, Element)

Description

Processes the XHTML string and inserts it into the specified Word document as formatted text after the given WordWriter Element using the given NamedStyle. Returns the last Element inserted into the document.

Signature: C#

```
public SoftArtisans.OfficeWriter.WordWriter.Element InsertHTML(System.String xhtml,
SoftArtisans.OfficeWriter.WordWriter.Document doc, SoftArtisans.OfficeWriter.WordWriter.NamedStyle
style, SoftArtisans.OfficeWriter.WordWriter.Element insertAfterElement)
```

Signature: VB.NET

```
Public Function InsertHTML(ByVal xhtml As String, ByVal doc As
SoftArtisans.OfficeWriter.WordWriter.Document, ByVal style As
SoftArtisans.OfficeWriter.WordWriter.NamedStyle, ByVal insertAfterElement As
SoftArtisans.OfficeWriter.WordWriter.Element) As SoftArtisans.OfficeWriter.WordWriter.Element
```

Parameters

xhtml

A well-formed fragment of an xhtml document.

doc

a WordWriter document object into which content will be inserted.

style

(Optional) a user-specified WordWriter style that contains the font to use when inserting text.

insertAfterElement

The WordWriter element after which the contents of the xhtml will be inserted.

Returns

The last Element inserted into the document.

Examples

C#

```
//--- Create a new WordWriter document.
WordApplication wApp = new WordApplication();
Document doc = wApp.Create();

//--- Create the HTMLToWord object.
HTMLToWord h2w = new HTMLToWord();

//--- The XHTML to insert into the WordWriter document.
string xhtml = "<h1>Greetings</h1><p><b>Hello</b>, <i>world</i>!</p>";

//--- Create a style that uses a red Arial 11 point font.
NamedStyle myStyle = doc.Styles[NamedStyle.BuiltIn.Normal];
Font myFont = myStyle.Font;
myFont.Name = "Arial";
myFont.Size = 11;
myFont.Color = System.Drawing.Color.Red;
myStyle.Font = myFont;

//--- Insert the XHTML at the end of the WordWriter document.
Element lastInsertedElement = h2w.InsertHTML(xhtml, doc, myStyle, table);
```

vb.net

```
'--- Create a new WordWriter document.
Dim wApp As New WordApplication()
Dim doc As Document = wApp.Create()

'--- Create the HTMLToWord object.
Dim h2w As New HTMLToWord()

'--- The XHTML to insert into the WordWriter document.
Dim xhtml As String = "<h1>Greetings</h1><p><b>Hello</b>, <i>world</i>!</p>"

'--- Insert the XHTML at the end of the WordWriter document.
Dim lastInsertedElement As Element = h2w.InsertHTML(xhtml, doc, null, doc)
```

HTMLToWord.InsertHTML(String, Document)

Description

Processes the XHTML string and inserts it at the end of the specified Word document as formatted text using the document's Normal style as the default font. Returns the last Element inserted into the document.

Signature: C#

```
public SoftArtisans.OfficeWriter.WordWriter.Element InsertHTML(System.String xhtml,
SoftArtisans.OfficeWriter.WordWriter.Document doc)
```

Signature: VB.NET

```
Public Function InsertHTML(ByVal xhtml As String, ByVal doc As
SoftArtisans.OfficeWriter.WordWriter.Document) As SoftArtisans.OfficeWriter.WordWriter.Element
```

Parameters

xhtml

A well-formed fragment of an xhtml document.

doc

a WordWriter document object into which content will be inserted.

Returns

The last Element inserted into the document.

Examples

C#

```
//--- Create a new WordWriter document.
 WordApplication wApp = new WordApplication();
 Document doc = wApp.Create();

 //--- Create the HTMLToWord object.
 HTMLToWord h2w = new HTMLToWord();

 //--- The XHTML to insert into the WordWriter document.
 string xhtml = "&lt;h1>Greetings&lt;/h1>&lt;p>&lt;b>Hello&lt;/b>, &lt;i>world&lt;/i>!&lt;/p>";

 //--- Insert the XHTML at the end of the WordWriter document.
 Element lastInsertedElement = h2w.InsertHTML(xhtml, doc);
```

vb.net

```
'--- Create a new WordWriter document.
 Dim wApp As New WordApplication()
 Dim doc As Document = wApp.Create()

 '--- Create the HTMLToWord object.
 Dim h2w As New HTMLToWord()

 '--- The XHTML to insert into the WordWriter document.
 Dim xhtml As String = "&lt;h1>Greetings&lt;/h1>&lt;p>&lt;b>Hello&lt;/b>, &lt;i>world&lt;/i>!&lt;/p>"

 '--- Insert the XHTML at the end of the WordWriter document.
 Dim lastInsertedElement As Element = h2w.InsertHTML(xhtml, doc)
```

HTMLToWord.InsertProperties

Description

Contains settings that control how unsupported HTML tags are treated and provides an override to the default font used when inserting text into the document.

Signature: C#

```
public HTMLInsertProperties InsertProperties{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property InsertProperties() As HTMLInsertProperties
```

Examples

C#

```
//--- Get the current settings.
 HTMLToWord.HTMLInsertProperties htmlProps = h2w.InsertProperties;

 //--- Tell HTMLToWord to skip XHTML tags that it does not explicitly handle.
 h2w.InsertProperties.InsertContentsOfUnknownTags = false;
```

vb.net

```
'--- Get the current settings.
 Dim htmlProps As HTMLToWord.HTMLInsertProperties = h2w.InsertProperties

 '--- Tell HTMLToWord to skip XHTML tags that it does not explicitly handle.
 h2w.InsertProperties.InsertContentsOfUnknownTags = False
```

HTMLToWord.LogFileName

Description

The full path to the HTMLToWord debug log file. To create a log file, you must set the DebugLogLevel parameter to a value other than None.

Signature: C#

```
public System.String LogFileName{ get; set; }
```

Signature: VB.NET

```
Public Property LogFileName() As String
```

Examples

C#

```
h2w.LogFileName = @"c:\SoftArtisans\HTMLProjects\Log Files";
```

vb.net

```
h2w.LogFileName = "c:\SoftArtisans\HTMLProjects\Log Files"
```

HTMLToWord.LoggingLevel

Description

A LoggingLevel value specified the amount of information written to the log file. To create a log file, you must also specify the full path for the file using the LogFileName property.

Signature: C#

```
public sealed enum LoggingLevel
```

Signature: VB.NET

```
Public NotInheritable Enum LoggingLevel
 End Enum
```

Values

Name	Description
Debug	Write error messages and additional debugging information to the log file.
Error	Write only error messages to the log file.
None	Do not write to the log file.

Trace	In addition to error and debugging messages, log the XHTML tags as they are parsed and the Elements as they are inserted into a document.

The HTMLToWord Object

Use the HTMLToWord object to insert well-formed HTML (XHTML) text into a WordWriter Documentobject. You can specify a style to use when inserting text or use the document's default style. You can also specify whether to insert text after a specific element or at the end of a document.

The contents of HTML tags not listed on the supported tags page may be inserted into the document as plain text. The behavior is dependent on the settings of the AcceptUnknownTags and IncludeContentsOfUnknownTags properties found in the HTMLInsertProperties property container.

For an overview of how to use the HTMLToWord object, see Using HTMLToWord.

Examples:

C#

```
//--- Well-formed HTML fragment to be inserted into the document.
string htmlFragment = @"<p><b>Hello,</b><i> World</i>!</p>";

//--- Open an existing Word document with the SoftArtisans WordApplication object.
WordApplication wApp = new WordApplication();
Document doc = wApp.Open("myDoument.doc");

//--- Create the HTMLToWord object.
HTMLToWord h2w = new HTMLToWord();

//--- Insert the HTML fragment at the end of the document.
//--- Use the default style.
h2w.InsertHTML(htmlFragment, doc, null, doc);
```

VB.NET

```
'--- Well-formed HTML fragment to be inserted into the document.
Dim htmlFragment As String = "<p><b>Hello,</b><i> World</i>!</p>"

'--- Open an existing Word document with the SoftArtisans WordApplication object.
Dim wApp As WordApplication = new WordApplication
Dim doc As Document = wApp.Open("myDoument.doc")

'--- Create the HTMLToWord object.
Dim h2w As HTMLToWord = new HTMLToWord

'--- Insert the HTML fragment at the end of the document.
'--- Use the default style.
h2w.InsertHTML(htmlFragment, doc, Nothing, doc)
```

HTMLToWord Constants

Name	Description
HTMLToWord.Delegates.InsertElementDelegate	Type for user-defined delegate method called by HTMLToWord before inserting an element. HTMLTagAction InsertElementDelegate(Document doc, Element insertAfterElement, XmlNode node)
HTMLToWord.Delegates.FormatElementDelegate	Type for user-defined delegate method called by HTMLToWord after inserting an element. void FormatElementDelegate(Element insertedElement, XmlNode node)
HTMLToWord.Delegates.GetImageDelegate	Type for user-defined delegate method called by HTMLToWord when an HTML img tag is encountered. System.IO.Stream GetImageDelegate(string URI)

HTMLToWord Enumerations

Name	Description
HTMLToWord.HTMLTagAction	Return value from the user-supplied InsertElementDelegate delegate method
HTMLToWord.LoggingLevel	A LoggingLevel value specified the amount of information written to the log file

HTMLToWord Properties

Property	Type	Access	Description
DebugLogLevel	HTMLToWord.LoggingLevel	read write	Sets the amount of information written to the debug log file
FormatDelegate	FormatElementDelegate	read write	User-assigned delegate method that is called after HTMLToWord inserts an Element into the Word document
ImageDelegate	GetImageDelegate	read write	User-assigned delegate method that allows users to provide an implementation for retrieving the contents of an image and passing it back to HTMLToWord for insertion into a document
InsertDelegate	InsertElementDelegate	read write	A user-assigned delegate method that HTMLToWord calls before it processes an XHTML tag
InsertProperties	HTMLInsertProperties	read write	Contains settings that control how unsupported HTML tags are treated and provides an override to the default font used when inserting text into the document.
LogFileName	String	read write	The full path to the HTMLToWord debug log file

HTMLToWord Methods

Returns	Signature and Description
Element	HTMLToWord.InsertHTML(String, Document) Processes the XHTML string and inserts it at the end of the specified Word document as formatted text using the document's Normal style as the default font. Returns the last Element inserted into the document
Element	HTMLToWord.InsertHTML(String, Document, Element) Processes the XHTML string and inserts it into the specified Word document as formatted text after the given WordWriter Element using the document's Normal style as the default font. Returns the last Element inserted into the document
Element	HTMLToWord.InsertHTML(String, Document, NamedStyle, Element) Processes the XHTML string and inserts it into the specified Word document as formatted text after the given WordWriter Element using the given NamedStyle. Returns the last Element inserted into the document

OfficeWriter Links

Documentation for other parts of OfficeWriter

In addition to WordWriter, OfficeWriter includes the following products:

- **ExcelWriter**
  - **ExcelWriter** allows you to create Excel spreadsheets on the server programmatically or from a template.
- **Reporting Services Integration**
  - **OfficeWriter Reporting Services Integration** allows you to render SQL Server Reporting Services reports according to templates using custom OfficeWriter renderers. Using the OfficeWriter Designer plug-in, end users can create or edit reports directly from Excel or Word and render them using the OfficeWriter renderers.
- **ExcelWriter COM**
  - **ExcelWriter COM** allows you to create Excel documents on the server programmatically or from a template using ASP.

ExcelWriter COM is no longer included in the standard install for OfficeWriter. If you need ExcelWriter COM, please contact SoftArtisans directly.

Online samples

There are extensive online samples available with the option to run and view code:

http://windemo.softartisans.com/OfficeWriter/latest/aspnet.aspx

External Links

- OfficeWriter Online
- OfficeWriter Forums
- SoftArtisans Technical Services
- SoftArtisans
- General Questions

WordTemplate 2003 feature limitation

WordWriter templates can be created in Windows 2003, however, if a template contains certain new Word 2003 features, WordWriter may throw an exception. This issue only applies to the creation of templates; WordWriter-generated output files can be viewed in Word 2003 with no limitations.