1. ExcelWriter for Windows Version 7 . . . 14
1.1 Introduction . . . 14
1.1.1 What Is ExcelWriter? . . . 14
1.1.2 Features and Benefits . . . 15
1.1.3 ExcelWriter Editions . . . 15
1.1.4 New Features . . . 16
1.1.5 Requirements . . . 17
1.1.6 Office 2007 Support . . . 18
1.1.7 Frequently Asked Questions . . . 18
1.1.8 Change Log . . . 19
1.2 Quick Start . . . 22
1.2.1 Create a Spreadsheet with ExcelTemplate . . . 22
1.2.2 Create a Spreadsheet with ExcelApplication . . . 25
1.2.3 Upgrading from ExcelWriter Version 6 to ExcelWriter Version 7 . . . 27
1.3 ExcelTemplate in Depth . . . 29
1.3.1 How to Use Templates . . . 30
1.3.2 Creating Data Markers . . . 32
1.3.3 Using an Array as a Data Source . . . 36
1.3.4 Using a Database as a Data Source . . . 40
1.3.5 Using Charts with ExcelTemplate . . . 43
1.3.6 Using Grouping and Nesting with ExcelTemplate . . . 47
1.3.7 Advanced Grouping and Nesting . . . 52
1.3.8 Creating a Grouped and Nested Spreadsheet . . . 59
1.3.9 ExcelTemplate Output Options . . . 65
1.3.10 Passing ExcelTemplate to ExcelApplication . . . 67
1.3.11 Templates and Pivot Tables . . . 68
1.3.11.1 Templates and Pivot Tables in Older Versions . . . 71
1.3.12 Stretching Formulas . . . 75
1.3.13 Drawing Objects and Data Markers . . . 76
1.4 ExcelApplication in Depth . . . 77
1.4.1 Addressing Cells . . . 77
1.4.2 Formulas . . . 79
1.4.3 Styles in ExcelWriter . . . 80
1.4.4 Number Formats . . . 82
1.4.5 The Color Palette . . . 83
1.4.6 Areas and Ranges . . . 85
1.4.7 Modifying an Existing Spreadsheet . . . 88
1.4.8 Importing Data . . . 89
1.4.9 Charts in ExcelApplication . . . 91
1.4.10 Drawing Objects . . . 95
1.4.11 Output Options for ExcelApplication . . . 96
1.4.12 Passing ExcelApplication to ExcelTemplate . . . 98
1.4.13 Formatting Headers and Footers . . . 100
1.4.14 Protecting Your Worksheet . . . 102
1.4.15 Copying Data From a Worksheet . . . 104
1.4.16 Multilingual Support . . . 106
1.5 SharePoint Integration in Depth . . . 107
1.5.1 Exporting list as Excel spreadsheet . . . 107
1.5.2 SharePoint Solution - Excel Export Plus . . . 117
1.6 Installation . . . 120
1.6.1 Installing ExcelWriter . . . 120
1.6.2 Removing ExcelWriter . . . 122
1.6.3 Security Considerations . . . 123
1.7 What is HotCell Technology? . . . 123
1.8 Programmer's Reference . . . 123
1.8.1 ExcelApplication Programmer's Reference . . . 124
1.8.1.1 Anchor . . . 126
1.8.1.1.1 Anchor.Column . . . 127
1.8.1.1.2 Anchor.OffsetX . . . 127
1.8.1.1.3 Anchor.OffsetY . . . 128
1.8.1.1.4 Anchor.Row . . . 128
1.8.1.2 Area . . . 129
1.8.1.2.1 Area.AllColumnWidths . . . 131
1.8.1.2.2 Area.AllColumnWidthsInChars . . . 132
1.8.1.2.3 Area.AllRowsHeight . . . 132
1.8.1.2.4 Area.ApplyStyle(Style) . . . 133
1.8.1.2.5 Area.ArrayFormula . . . 133
1.8.1.2.6 Area.AutoFitHeight() . . . 134
1.8.1.2.7 Area.AutoFitWidth() . . . 134
1.8.1.2.8 Area.BorderAround . . . 135
1.8.1.2.9 Area.ClearContent() . . . 135
1.8.1.2.10 Area.ColumnCount . . . 136
1.8.1.2.11 Area.CreateHyperlink(String) . . . 136
1.8.1.2.12 Area.Dimensions . . . 137
1.8.1.2.13 Area.FirstColumn . . . 137

1.8.1.2.14 Area.FirstRow . . . 138
1.8.1.2.15 Area.GetColumnWidth(Int32) . . . 138
1.8.1.2.16 Area.GetColumnWidthInChars(Int32) . . . 139
1.8.1.2.17 Area.GetRowHeight(Int32) . . . 140
1.8.1.2.18 Area.GroupColumns(Boolean) . . . 140
1.8.1.2.19 Area.GroupRows(Boolean) . . . 141
1.8.1.2.20 Area.ImportData . . . 141
1.8.1.2.21 Area.Item(Int32, Int32) . . . 149
1.8.1.2.22 Area.MergeCells() . . . 150
1.8.1.2.23 Area.PopulatedCells . . . 150
1.8.1.2.24 Area.RemoveConditionalFormat() . . . 151
1.8.1.2.25 Area.RowCount . . . 151
1.8.1.2.26 Area.SetColumnWidth(Int32, Double) . . . 152
1.8.1.2.27 Area.SetColumnWidthInChars(Int32, Double) . . . 152
1.8.1.2.28 Area.SetConditionalFormat(ConditionalFormat) . . . 153
1.8.1.2.29 Area.SetDataValidation(DataValidation) . . . 153
1.8.1.2.30 Area.SetRowHeight(Int32, Double) . . . 154
1.8.1.2.31 Area.SetStyle(Style) . . . 154
1.8.1.2.32 Area.ToString() . . . 155
1.8.1.2.33 Area.UngroupColumns() . . . 155
1.8.1.2.34 Area.UngroupRows() . . . 156
1.8.1.2.35 Area.WorksheetIndex . . . 156
1.8.1.3 Axis . . . 157
1.8.1.3.1 Axis.AxisLine . . . 157
1.8.1.3.2 Axis.LabelFont . . . 158
1.8.1.3.3 Axis.MajorGridline . . . 158
1.8.1.3.4 Axis.MajorTickType . . . 158
1.8.1.3.5 Axis.MinorGridline . . . 158
1.8.1.3.6 Axis.MinorTickType . . . 159
1.8.1.3.7 Axis.NumberFormat . . . 159
1.8.1.3.8 Axis.TextDisplayTopToBottom . . . 159
1.8.1.3.9 Axis.TextRotationAngle . . . 159
1.8.1.3.10 Axis.TickLabelPos . . . 160
1.8.1.3.11 Axis.TickLabelPosition . . . 160
1.8.1.3.12 Axis.TickType . . . 160
1.8.1.3.13 Axis.Title . . . 161
1.8.1.3.14 Axis.ValuesInReverseOrder . . . 161
1.8.1.3.15 Axis.Visible . . . 161
1.8.1.4 AxisLabel . . . 162
1.8.1.4.1 AxisLabel.LabelPosition . . . 162
1.8.1.4.2 AxisLabel.Position . . . 162
1.8.1.5 AxisType . . . 163
1.8.1.6 Border . . . 163
1.8.1.6.1 Border.Color . . . 164
1.8.1.6.2 Border.Item(Border.Part) . . . 165
1.8.1.6.3 Border.LineStyle . . . 165
1.8.1.6.4 Border.Part . . . 166
1.8.1.6.5 Border.Style . . . 167
1.8.1.7 BorderPart . . . 167
1.8.1.7.1 BorderPart.Color . . . 168
1.8.1.7.2 BorderPart.Style . . . 169
1.8.1.8 CategoryAxis . . . 169
1.8.1.8.1 CategoryAxis.AxisCrossesMaxCategory . . . 170
1.8.1.8.2 CategoryAxis.AxisCrossesMidCategory . . . 170
1.8.1.8.3 CategoryAxis.AxisDisplayType . . . 171
1.8.1.8.4 CategoryAxis.AxisTimeUnit . . . 171
1.8.1.8.5 CategoryAxis.BaseUnit . . . 172
1.8.1.8.6 CategoryAxis.CategoriesBetweenTickMarkLabels . . . 172
1.8.1.8.7 CategoryAxis.CategoriesBetweenTickMarks . . . 172
1.8.1.8.8 CategoryAxis.DisplayType . . . 172
1.8.1.8.9 CategoryAxis.MajorUnit . . . 173
1.8.1.8.10 CategoryAxis.MajorUnitValue . . . 173
1.8.1.8.11 CategoryAxis.MaximumValue . . . 173
1.8.1.8.12 CategoryAxis.MaximumValueAsDate . . . 173
1.8.1.8.13 CategoryAxis.MinimumValue . . . 173
1.8.1.8.14 CategoryAxis.MinimumValueAsDate . . . 174
1.8.1.8.15 CategoryAxis.MinorUnit . . . 174
1.8.1.8.16 CategoryAxis.MinorUnitValue . . . 174
1.8.1.8.17 CategoryAxis.ValueCrossingPoint . . . 174
1.8.1.8.18 CategoryAxis.ValueCrossingPointAsDate . . . 175
1.8.1.9 Cell . . . 175
1.8.1.9.1 Cell.ApplyStyle(Style) . . . 177
1.8.1.9.2 Cell.CellValueType . . . 177
1.8.1.9.3 Cell.ClearContent() . . . 178
1.8.1.9.4 Cell.ClearRichTextFormatting() . . . 178

1.8.1.9.5 Cell.ColumnNumber . . . 179
1.8.1.9.6 Cell.Comment . . . 179
1.8.1.9.7 Cell.ConditionalFormat . . . 180
1.8.1.9.8 Cell.CreateAnchor(Int32, Int32) . . . 180
1.8.1.9.9 Cell.CreateHyperlink(String) . . . 181
1.8.1.9.10 Cell.DataValidation . . . 181
1.8.1.9.11 Cell.Formula . . . 182
1.8.1.9.12 Cell.GetCharacters . . . 182
1.8.1.9.13 Cell.HasComment . . . 184
1.8.1.9.14 Cell.HasConditionalFormat . . . 184
1.8.1.9.15 Cell.HasDataValidation . . . 185
1.8.1.9.16 Cell.IsMerged . . . 185
1.8.1.9.17 Cell.Name . . . 186
1.8.1.9.18 Cell.RemoveComment() . . . 186
1.8.1.9.19 Cell.RemoveConditionalFormat() . . . 186
1.8.1.9.20 Cell.RemoveDataValidation() . . . 187
1.8.1.9.21 Cell.RowNumber . . . 187
1.8.1.9.22 Cell.SetAddInFormula(String) . . . 188
1.8.1.9.23 Cell.SetConditionalFormat(ConditionalFormat) . . . 188
1.8.1.9.24 Cell.Style . . . 189
1.8.1.9.25 Cell.ToString() . . . 190
1.8.1.9.26 Cell.Unmerge() . . . 190
1.8.1.9.27 Cell.Value . . . 191
1.8.1.9.28 Cell.ValueType . . . 191
1.8.1.10 Cells . . . 192
1.8.1.10.1 Cells.Item . . . 192
1.8.1.11 CellStyle . . . 194
1.8.1.12 CharacterRun . . . 194
1.8.1.12.1 CharacterRun.Clear() . . . 195
1.8.1.12.2 CharacterRun.Font . . . 195
1.8.1.12.3 CharacterRun.Length . . . 196
1.8.1.12.4 CharacterRun.StartIndex . . . 196
1.8.1.13 Chart . . . 197
1.8.1.13.1 Chart.CategoryAxis . . . 198
1.8.1.13.2 Chart.ChartArea . . . 198
1.8.1.13.3 Chart.ChartLocation . . . 199
1.8.1.13.4 Chart.ChartSettings3D . . . 199
1.8.1.13.5 Chart.ChartType . . . 199
1.8.1.13.6 Chart.DataTable . . . 199
1.8.1.13.7 Chart.Height . . . 200
1.8.1.13.8 Chart.Legend . . . 200
1.8.1.13.9 Chart.MoveChart . . . 200
1.8.1.13.10 Chart.Name . . . 202
1.8.1.13.11 Chart.ObjectPositioning . . . 202
1.8.1.13.12 Chart.PageSetup . . . 203
1.8.1.13.13 Chart.PlotArea . . . 203
1.8.1.13.14 Chart.Position . . . 203
1.8.1.13.15 Chart.PositionInCollection . . . 203
1.8.1.13.16 Chart.ScatterValueAxis . . . 204
1.8.1.13.17 Chart.SecondaryCategoryAxis . . . 204
1.8.1.13.18 Chart.SecondaryValueAxis . . . 204
1.8.1.13.19 Chart.SeriesCollection . . . 204
1.8.1.13.20 Chart.ShowRoundedCorners . . . 205
1.8.1.13.21 Chart.Title . . . 205
1.8.1.13.22 Chart.ValueAxis . . . 205
1.8.1.13.23 Chart.Width . . . 205
1.8.1.13.24 Chart.ZCategoryAxis . . . 206
1.8.1.14 ChartDataTable . . . 206
1.8.1.14.1 ChartDataTable.BorderLine . . . 207
1.8.1.14.2 ChartDataTable.Font . . . 207
1.8.1.14.3 ChartDataTable.Height . . . 207
1.8.1.14.4 ChartDataTable.ShowHorizontalBorders . . . 207
1.8.1.14.5 ChartDataTable.ShowLegendKeys . . . 208
1.8.1.14.6 ChartDataTable.ShowOutlineBorders . . . 208
1.8.1.14.7 ChartDataTable.ShowVerticalBorders . . . 208
1.8.1.14.8 ChartDataTable.Visible . . . 208
1.8.1.14.9 ChartDataTable.Width . . . 209
1.8.1.14.10 ChartDataTable.X . . . 209
1.8.1.14.11 ChartDataTable.Y . . . 209
1.8.1.15 ChartFillPattern . . . 209
1.8.1.16 ChartInterior . . . 212
1.8.1.16.1 ChartInterior.BackgroundColor . . . 212
1.8.1.16.2 ChartInterior.ClearFillEffects() . . . 213
1.8.1.16.3 ChartInterior.ForegroundColor . . . 213
1.8.1.16.4 ChartInterior.Pattern . . . 213

1.8.1.16.5 ChartInterior.Visible . . . 213
1.8.1.17 ChartLine . . . 214
1.8.1.17.1 ChartLine.Color . . . 214
1.8.1.17.2 ChartLine.LineStyle . . . 214
1.8.1.17.3 ChartLine.LineWeight . . . 215
1.8.1.17.4 ChartLine.Style . . . 215
1.8.1.17.5 ChartLine.Visible . . . 216
1.8.1.17.6 ChartLine.Weight . . . 216
1.8.1.18 ChartPageSetup . . . 216
1.8.1.18.1 ChartPageSetup.ChartSize . . . 217
1.8.1.18.2 ChartPageSetup.PrintedChartSize . . . 217
1.8.1.19 ChartRegion . . . 218
1.8.1.19.1 ChartRegion.BorderLine . . . 218
1.8.1.19.2 ChartRegion.Font . . . 219
1.8.1.19.3 ChartRegion.HasShadow . . . 219
1.8.1.19.4 ChartRegion.Height . . . 219
1.8.1.19.5 ChartRegion.HorizontalAlignment . . . 219
1.8.1.19.6 ChartRegion.Interior . . . 220
1.8.1.19.7 ChartRegion.TextDisplayTopToBottom . . . 220
1.8.1.19.8 ChartRegion.TextHorizontalAlignment . . . 220
1.8.1.19.9 ChartRegion.TextRotationAngle . . . 221
1.8.1.19.10 ChartRegion.TextVerticalAlignment . . . 221
1.8.1.19.11 ChartRegion.VerticalAlignment . . . 221
1.8.1.19.12 ChartRegion.Width . . . 222
1.8.1.19.13 ChartRegion.X . . . 222
1.8.1.19.14 ChartRegion.Y . . . 222
1.8.1.20 Charts . . . 222
1.8.1.20.1 Charts.AddChart(Chart, Anchor) . . . 223
1.8.1.20.2 Charts.Count . . . 224
1.8.1.20.3 Charts.CreateChart(ChartType, Anchor) . . . 224
1.8.1.20.4 Charts.GetEnumerator() . . . 224
1.8.1.20.5 Charts.Item . . . 225
1.8.1.20.6 Charts.Remove . . . 226
1.8.1.21 ChartSettings3D . . . 226
1.8.1.21.1 ChartSettings3D.ChartDepth . . . 227
1.8.1.21.2 ChartSettings3D.DistanceFromEyeToChart . . . 227
1.8.1.21.3 ChartSettings3D.ElevationAngle . . . 228
1.8.1.21.4 ChartSettings3D.Floor3D . . . 228
1.8.1.21.5 ChartSettings3D.Floor3DBorder . . . 228
1.8.1.21.6 ChartSettings3D.PlotVolumeHeight . . . 228
1.8.1.21.7 ChartSettings3D.RotationAngle . . . 229
1.8.1.21.8 ChartSettings3D.SeriesGapWidth . . . 229
1.8.1.21.9 ChartSettings3D.UseAutoScaling . . . 229
1.8.1.21.10 ChartSettings3D.UseRightAngleAxes . . . 229
1.8.1.21.11 ChartSettings3D.Wall3D . . . 230
1.8.1.21.12 ChartSettings3D.Wall3DBorder . . . 230
1.8.1.22 Chartsheet . . . 230
1.8.1.22.1 Chartsheet.Chart . . . 231
1.8.1.23 ChartText . . . 232
1.8.1.23.1 ChartText.ClearRichTextFormatting() . . . 232
1.8.1.23.2 ChartText.Formula . . . 233
1.8.1.23.3 ChartText.GetCharacters . . . 233
1.8.1.23.4 ChartText.NumberFormat . . . 234
1.8.1.23.5 ChartText.Text . . . 234
1.8.1.24 ChartTextCollection . . . 234
1.8.1.24.1 ChartTextCollection.AddChartText(ChartText) . . . 235
1.8.1.24.2 ChartTextCollection.Count . . . 235
1.8.1.24.3 ChartTextCollection.CreateChartText . . . 236
1.8.1.24.4 ChartTextCollection.Item(Int32) . . . 237
1.8.1.24.5 ChartTextCollection.Remove . . . 237
1.8.1.25 ChartType . . . 238
1.8.1.25.1 ChartType.Area . . . 239
1.8.1.25.2 ChartType.Bar . . . 240
1.8.1.25.3 ChartType.Bubble . . . 241
1.8.1.25.4 ChartType.Category . . . 241
1.8.1.25.5 ChartType.ChartCategory . . . 242
1.8.1.25.6 ChartType.Column . . . 242
1.8.1.25.7 ChartType.Custom . . . 244
1.8.1.25.8 ChartType.Doughnut . . . 244
1.8.1.25.9 ChartType.Is3D . . . 245
1.8.1.25.10 ChartType.Line . . . 245
1.8.1.25.11 ChartType.Name . . . 246
1.8.1.25.12 ChartType.Pie . . . 246
1.8.1.25.13 ChartType.Scatter . . . 247
1.8.1.25.14 ChartType.Surface . . . 248

1.8.1.26 Color . . . 248
1.8.1.26.1 Color.Blue . . . 250
1.8.1.26.2 Color.Green . . . 250
1.8.1.26.3 Color.IsCustomColor . . . 250
1.8.1.26.4 Color.IsPaletteColor . . . 251
1.8.1.26.5 Color.IsSystemColor . . . 251
1.8.1.26.6 Color.PaletteIndex . . . 252
1.8.1.26.7 Color.Red . . . 252
1.8.1.26.8 Color.SystemColor . . . 252
1.8.1.26.9 Color.ToString() . . . 253
1.8.1.27 ColumnProperties . . . 253
1.8.1.27.1 ColumnProperties.ApplyStyle(Style) . . . 255
1.8.1.27.2 ColumnProperties.AutoFitWidth() . . . 255
1.8.1.27.3 ColumnProperties.ColumnNumber . . . 255
1.8.1.27.4 ColumnProperties.Hidden . . . 256
1.8.1.27.5 ColumnProperties.OutlineCollapsed . . . 256
1.8.1.27.6 ColumnProperties.OutlineLevel . . . 257
1.8.1.27.7 ColumnProperties.PaddingInChars . . . 257
1.8.1.27.8 ColumnProperties.Style . . . 258
1.8.1.27.9 ColumnProperties.Width . . . 258
1.8.1.27.10 ColumnProperties.WidthInChars . . . 259
1.8.1.28 Comment . . . 259
1.8.1.28.1 Comment.Author . . . 260
1.8.1.28.2 Comment.Cell . . . 260
1.8.1.28.3 Comment.ClearRichTextFormatting() . . . 260
1.8.1.28.4 Comment.FitToText . . . 261
1.8.1.28.5 Comment.GetCharacters . . . 261
1.8.1.28.6 Comment.HorizontalAlignment . . . 262
1.8.1.28.7 Comment.Shape . . . 262
1.8.1.28.8 Comment.Text . . . 262
1.8.1.28.9 Comment.VerticalAlignment . . . 263
1.8.1.28.10 Comment.Visible . . . 263
1.8.1.29 Comments . . . 263
1.8.1.29.1 Comments.Count . . . 264
1.8.1.29.2 Comments.CreateComment . . . 264
1.8.1.29.3 Comments.GetEnumerator() . . . 265
1.8.1.29.4 Comments.Item . . . 265
1.8.1.29.5 Comments.Remove(Int32) . . . 266
1.8.1.30 Condition . . . 267
1.8.1.30.1 Condition.ApplyStyle(Style) . . . 268
1.8.1.30.2 Condition.ChangeCondition . . . 269
1.8.1.30.3 Condition.Comparison . . . 271
1.8.1.30.4 Condition.ComparisonType . . . 272
1.8.1.30.5 Condition.Formula1 . . . 272
1.8.1.30.6 Condition.Formula2 . . . 273
1.8.1.30.7 Condition.Style . . . 274
1.8.1.31 ConditionalFormat . . . 275
1.8.1.31.1 ConditionalFormat.Count . . . 276
1.8.1.31.2 ConditionalFormat.CreateCondition . . . 276
1.8.1.31.3 ConditionalFormat.Item(Int32) . . . 278
1.8.1.31.4 ConditionalFormat.Range . . . 279
1.8.1.31.5 ConditionalFormat.RemoveCondition . . . 280
1.8.1.32 CopyPasteProperties . . . 282
1.8.1.32.1 CopyPasteProperties.CopyColumnWidth . . . 283
1.8.1.32.2 CopyPasteProperties.CopyComments . . . 284
1.8.1.32.3 CopyPasteProperties.CopyFormatting . . . 284
1.8.1.32.4 CopyPasteProperties.CopyFormulas . . . 285
1.8.1.32.5 CopyPasteProperties.CopyMergedCells . . . 286
1.8.1.32.6 CopyPasteProperties.CopyPasteType . . . 286
1.8.1.32.7 CopyPasteProperties.CopyRowHeight . . . 287
1.8.1.32.8 CopyPasteProperties.CopyValues . . . 287
1.8.1.33 DataImportProperties . . . 288
1.8.1.33.1 DataImportProperties.ALLCOLUMNS . . . 289
1.8.1.33.2 DataImportProperties.ALLROWS . . . 289
1.8.1.33.3 DataImportProperties.ColumnIndexFilter . . . 290
1.8.1.33.4 DataImportProperties.ColumnNameFilter . . . 290
1.8.1.33.5 DataImportProperties.IgnoreNulls . . . 291
1.8.1.33.6 DataImportProperties.MaxColumns . . . 291
1.8.1.33.7 DataImportProperties.MaxRows . . . 292
1.8.1.33.8 DataImportProperties.Transpose . . . 292
1.8.1.33.9 DataImportProperties.Truncate . . . 293
1.8.1.33.10 DataImportProperties.TruncateColumns . . . 294
1.8.1.33.11 DataImportProperties.TruncateRows . . . 295
1.8.1.33.12 DataImportProperties.UseColumnNames . . . 295
1.8.1.34 DataLabelPlacementType . . . 296

1.8.1.35 DataPoint . . . 297
1.8.1.35.1 DataPoint.DistanceFromCenter . . . 298
1.8.1.35.2 DataPoint.Interior . . . 298
1.8.1.35.3 DataPoint.IsOnSecondaryPieorBar . . . 298
1.8.1.35.4 DataPoint.Label . . . 298
1.8.1.35.5 DataPoint.LegendEntryFont . . . 299
1.8.1.35.6 DataPoint.Line . . . 299
1.8.1.35.7 DataPoint.Marker . . . 299
1.8.1.35.8 DataPoint.ShowLegendEntry . . . 299
1.8.1.36 DataPointMarker . . . 300
1.8.1.36.1 DataPointMarker.BackgroundColor . . . 300
1.8.1.36.2 DataPointMarker.ForegroundColor . . . 301
1.8.1.36.3 DataPointMarker.HasShadow . . . 301
1.8.1.36.4 DataPointMarker.MarkerType . . . 301
1.8.1.36.5 DataPointMarker.ShapeType . . . 301
1.8.1.36.6 DataPointMarker.ShowBackgroundColor . . . 302
1.8.1.36.7 DataPointMarker.ShowForegroundColor . . . 302
1.8.1.36.8 DataPointMarker.Size . . . 302
1.8.1.36.9 DataPointMarker.Visible . . . 303
1.8.1.37 DataValidation . . . 303
1.8.1.37.1 DataValidation.AllowedType . . . 305
1.8.1.37.2 DataValidation.Clear() . . . 305
1.8.1.37.3 DataValidation.Comparison . . . 305
1.8.1.37.4 DataValidation.ComparisonType . . . 305
1.8.1.37.5 DataValidation.ErrorAlert . . . 306
1.8.1.37.6 DataValidation.ErrorAlertStyle . . . 306
1.8.1.37.7 DataValidation.ErrorAlertStyleType . . . 306
1.8.1.37.8 DataValidation.ErrorAlertTitle . . . 307
1.8.1.37.9 DataValidation.IgnoreBlanks . . . 307
1.8.1.37.10 DataValidation.InputMessage . . . 307
1.8.1.37.11 DataValidation.MaximumValue . . . 308
1.8.1.37.12 DataValidation.MinimumValue . . . 308
1.8.1.37.13 DataValidation.SetAllowedType . . . 308
1.8.1.37.14 DataValidation.ShowErrorAlert . . . 310
1.8.1.37.15 DataValidation.ShowInputMessage . . . 310
1.8.1.37.16 DataValidation.Title . . . 310
1.8.1.37.17 DataValidation.UseListDropdown . . . 311
1.8.1.37.18 DataValidation.ValidationType . . . 311
1.8.1.38 DocumentProperties . . . 311
1.8.1.38.1 DocumentProperties.Application . . . 313
1.8.1.38.2 DocumentProperties.Author . . . 313
1.8.1.38.3 DocumentProperties.BuiltIn . . . 314
1.8.1.38.4 DocumentProperties.Category . . . 315
1.8.1.38.5 DocumentProperties.Clear() . . . 315
1.8.1.38.6 DocumentProperties.Comments . . . 316
1.8.1.38.7 DocumentProperties.Company . . . 316
1.8.1.38.8 DocumentProperties.CreationDate . . . 317
1.8.1.38.9 DocumentProperties.GetCustomProperty(String) . . . 317
1.8.1.38.10 DocumentProperties.Item . . . 318
1.8.1.38.11 DocumentProperties.Keywords . . . 320
1.8.1.38.12 DocumentProperties.LastSavedDate . . . 320
1.8.1.38.13 DocumentProperties.Manager . . . 321
1.8.1.38.14 DocumentProperties.RestoreDefaults() . . . 321
1.8.1.38.15 DocumentProperties.SetCustomProperty(String, Object) . . . 322
1.8.1.38.16 DocumentProperties.Title . . . 322
1.8.1.39 ErrorBar . . . 323
1.8.1.39.1 ErrorBar.BarDirection . . . 324
1.8.1.39.2 ErrorBar.CustomMinusFormula . . . 324
1.8.1.39.3 ErrorBar.CustomPlusFormula . . . 324
1.8.1.39.4 ErrorBar.Direction . . . 325
1.8.1.39.5 ErrorBar.ErrorValueType . . . 325
1.8.1.39.6 ErrorBar.Line . . . 325
1.8.1.39.7 ErrorBar.TShaped . . . 325
1.8.1.39.8 ErrorBar.Value . . . 326
1.8.1.39.9 ErrorBar.ValueType . . . 326
1.8.1.40 ExcelApplication . . . 326
1.8.1.40.1 ExcelApplication.Create() . . . 328
1.8.1.40.2 ExcelApplication.ExcelApplication() . . . 328
1.8.1.40.3 ExcelApplication.LicenseKey . . . 329
1.8.1.40.4 ExcelApplication.Open . . . 329
1.8.1.40.5 ExcelApplication.Save . . . 334
1.8.1.40.6 ExcelApplication.Version . . . 337
1.8.1.41 ExcelWriterException . . . 338
1.8.1.42 Font . . . 338
1.8.1.42.1 Font.Bold . . . 339

1.8.1.42.2 Font.Color . . . 340
1.8.1.42.3 Font.Italic . . . 340
1.8.1.42.4 Font.Name . . . 341
1.8.1.42.5 Font.ScriptStyle . . . 341
1.8.1.42.6 Font.Size . . . 342
1.8.1.42.7 Font.Strikethrough . . . 342
1.8.1.42.8 Font.Underline . . . 343
1.8.1.42.9 Font.UnderlineStyle . . . 344
1.8.1.42.10 Font.VerticalPosition . . . 344
1.8.1.43 GlobalStyle . . . 345
1.8.1.44 Group . . . 345
1.8.1.44.1 Group.Height . . . 346
1.8.1.44.2 Group.Position . . . 346
1.8.1.44.3 Group.Type . . . 346
1.8.1.44.4 Group.Width . . . 347
1.8.1.45 Groups . . . 347
1.8.1.45.1 Groups.Count . . . 347
1.8.1.45.2 Groups.GetEnumerator() . . . 348
1.8.1.45.3 Groups.ListGroups() . . . 348
1.8.1.45.4 Groups.Remove . . . 348
1.8.1.46 HeaderFooterPicture . . . 349
1.8.1.46.1 HeaderFooterPicture.Height . . . 350
1.8.1.46.2 HeaderFooterPicture.LockAspectRatio . . . 351
1.8.1.46.3 HeaderFooterPicture.Width . . . 351
1.8.1.47 HeaderFooterSection . . . 352
1.8.1.47.1 HeaderFooterSection.GetPicture() . . . 353
1.8.1.47.2 HeaderFooterSection.GetText() . . . 353
1.8.1.47.3 HeaderFooterSection.Section . . . 353
1.8.1.47.4 HeaderFooterSection.SetContent . . . 354
1.8.1.48 Hyperlink . . . 356
1.8.1.48.1 Hyperlink.Area . . . 357
1.8.1.48.2 Hyperlink.Href . . . 357
1.8.1.48.3 Hyperlink.Text . . . 358
1.8.1.49 Hyperlinks . . . 358
1.8.1.49.1 Hyperlinks.Count . . . 359
1.8.1.49.2 Hyperlinks.CreateHyperlink . . . 360
1.8.1.49.3 Hyperlinks.GetEnumerator() . . . 361
1.8.1.49.4 Hyperlinks.Item(Int32) . . . 361
1.8.1.50 Legend . . . 362
1.8.1.50.1 Legend.LegendLocation . . . 363
1.8.1.50.2 Legend.Location . . . 363
1.8.1.50.3 Legend.Vertical . . . 363
1.8.1.50.4 Legend.Visible . . . 364
1.8.1.51 NamedObject . . . 364
1.8.1.51.1 NamedObject.BuiltInName . . . 365
1.8.1.51.2 NamedObject.Formula . . . 366
1.8.1.51.3 NamedObject.Name . . . 366
1.8.1.51.4 NamedObject.NamedObjectType . . . 367
1.8.1.51.5 NamedObject.NameScope . . . 367
1.8.1.51.6 NamedObject.ReturnType . . . 367
1.8.1.51.7 NamedObject.Scope . . . 368
1.8.1.51.8 NamedObject.Value . . . 368
1.8.1.52 NamedStyle . . . 369
1.8.1.52.1 NamedStyle.Name . . . 370
1.8.1.53 NumberFormat . . . 371
1.8.1.53.1 NumberFormat.Color . . . 372
1.8.1.53.2 NumberFormat.CreateAccounting(Int32, Boolean, NumberFormat.Color) . . . 372
1.8.1.53.3 NumberFormat.CreateCurrency . . . 373
1.8.1.53.4 NumberFormat.CreateFraction . . . 376
1.8.1.53.5 NumberFormat.CreateNumber(Int32, Boolean, Boolean, Boolean, Boolean, NumberFormat.Color) . . . 378
1.8.1.53.6 NumberFormat.CreatePercentage(Int32) . . . 379
1.8.1.53.7 NumberFormat.CreateScientific(Int32) . . . 380
1.8.1.53.8 NumberFormat.DateFormat . . . 380
1.8.1.53.9 NumberFormat.Special_EN_US . . . 381
1.8.1.53.10 NumberFormat.TimeFormat . . . 382
1.8.1.54 ObjectPositioning . . . 382
1.8.1.55 Othersheet . . . 383
1.8.1.56 PageSetup . . . 383
1.8.1.56.1 PageSetup.BottomMargin . . . 385
1.8.1.56.2 PageSetup.CellErrorsAs . . . 386
1.8.1.56.3 PageSetup.CenterFooter . . . 386
1.8.1.56.4 PageSetup.CenterHeader . . . 387
1.8.1.56.5 PageSetup.CenterHorizontally . . . 387
1.8.1.56.6 PageSetup.CenterVertically . . . 388
1.8.1.56.7 PageSetup.FirstPageNumber . . . 389

1.8.1.56.8 PageSetup.FirstPageNumberAuto . . . 389
1.8.1.56.9 PageSetup.FitToPagesTall . . . 390
1.8.1.56.10 PageSetup.FitToPagesWide . . . 390
1.8.1.56.11 PageSetup.FooterMargin . . . 391
1.8.1.56.12 PageSetup.GetFooter(HeaderFooterSection.Section) . . . 392
1.8.1.56.13 PageSetup.GetHeader(HeaderFooterSection.Section) . . . 392
1.8.1.56.14 PageSetup.HeaderMargin . . . 392
1.8.1.56.15 PageSetup.LeftFooter . . . 393
1.8.1.56.16 PageSetup.LeftHeader . . . 394
1.8.1.56.17 PageSetup.LeftMargin . . . 394
1.8.1.56.18 PageSetup.Orientation . . . 395
1.8.1.56.19 PageSetup.PageOrientation . . . 396
1.8.1.56.20 PageSetup.PagePaperSize . . . 396
1.8.1.56.21 PageSetup.PagePrintOrder . . . 397
1.8.1.56.22 PageSetup.PaperSize . . . 398
1.8.1.56.23 PageSetup.PrintArea . . . 398
1.8.1.56.24 PageSetup.PrintAsDraft . . . 399
1.8.1.56.25 PageSetup.PrintCellErrorsAs . . . 399
1.8.1.56.26 PageSetup.PrintComments . . . 400
1.8.1.56.27 PageSetup.PrintCommentsAtEnd . . . 400
1.8.1.56.28 PageSetup.PrintGridlines . . . 401
1.8.1.56.29 PageSetup.PrintHeadings . . . 402
1.8.1.56.30 PageSetup.PrintInBlackAndWhite . . . 402
1.8.1.56.31 PageSetup.PrintOrder . . . 403
1.8.1.56.32 PageSetup.PrintTitles . . . 403
1.8.1.56.33 PageSetup.RightFooter . . . 404
1.8.1.56.34 PageSetup.RightHeader . . . 404
1.8.1.56.35 PageSetup.RightMargin . . . 405
1.8.1.56.36 PageSetup.SetPrintTitleColumns(Int32, Int32) . . . 406
1.8.1.56.37 PageSetup.SetPrintTitleRows(Int32, Int32) . . . 406
1.8.1.56.38 PageSetup.TopMargin . . . 407
1.8.1.56.39 PageSetup.UseZoom . . . 407
1.8.1.56.40 PageSetup.Zoom . . . 408
1.8.1.57 Palette . . . 408
1.8.1.57.1 Palette.GetClosestColor . . . 410
1.8.1.57.2 Palette.GetColor . . . 411
1.8.1.57.3 Palette.GetColorAt(Int32) . . . 412
1.8.1.57.4 Palette.MaxColorIndex . . . 413
1.8.1.57.5 Palette.MaxRGBValue . . . 413
1.8.1.57.6 Palette.MinColorIndex . . . 413
1.8.1.57.7 Palette.MinRGBValue . . . 414
1.8.1.57.8 Palette.SetColor(Color, Int32, Int32, Int32) . . . 414
1.8.1.57.9 Palette.SetColorAt(Int32, Int32, Int32, Int32) . . . 415
1.8.1.57.10 Palette.SystemColor . . . 415
1.8.1.58 Pattern . . . 416
1.8.1.59 Picture . . . 417
1.8.1.59.1 Picture.Height . . . 418
1.8.1.59.2 Picture.LockAspectRatio . . . 418
1.8.1.59.3 Picture.Name . . . 418
1.8.1.59.4 Picture.PictureData . . . 418
1.8.1.59.5 Picture.Position . . . 419
1.8.1.59.6 Picture.Shape . . . 419
1.8.1.59.7 Picture.Width . . . 419
1.8.1.60 Pictures . . . 419
1.8.1.60.1 Pictures.Count . . . 420
1.8.1.60.2 Pictures.CreatePicture . . . 420
1.8.1.60.3 Pictures.GetEnumerator() . . . 421
1.8.1.60.4 Pictures.Item . . . 422
1.8.1.60.5 Pictures.ListPictures() . . . 423
1.8.1.60.6 Pictures.Remove . . . 423
1.8.1.61 Range . . . 424
1.8.1.61.1 Range.ApplyStyle(Style) . . . 425
1.8.1.61.2 Range.AreaCount . . . 426
1.8.1.61.3 Range.Areas . . . 426
1.8.1.61.4 Range.BorderAround . . . 427
1.8.1.61.5 Range.ClearContent() . . . 427
1.8.1.61.6 Range.FirstCellStyle . . . 428
1.8.1.61.7 Range.Item(Int32) . . . 428
1.8.1.61.8 Range.JoinRange(Range) . . . 429
1.8.1.61.9 Range.RemoveConditionalFormat() . . . 429
1.8.1.61.10 Range.SetConditionalFormat(ConditionalFormat) . . . 430
1.8.1.61.11 Range.SetDataValidation(DataValidation) . . . 430
1.8.1.61.12 Range.SetStyle(Style) . . . 431
1.8.1.61.13 Range.ToString() . . . 431
1.8.1.62 RowProperties . . . 432

1.8.1.62.1 RowProperties.ApplyStyle(Style) . . . 433
1.8.1.62.2 RowProperties.AutoFitHeight() . . . 433
1.8.1.62.3 RowProperties.Height . . . 433
1.8.1.62.4 RowProperties.Hidden . . . 434
1.8.1.62.5 RowProperties.OutlineCollapsed . . . 435
1.8.1.62.6 RowProperties.OutlineLevel . . . 435
1.8.1.62.7 RowProperties.RowNumber . . . 436
1.8.1.62.8 RowProperties.Style . . . 436
1.8.1.63 Series . . . 437
1.8.1.63.1 Series.AxisType . . . 438
1.8.1.63.2 Series.BubbleSizes . . . 438
1.8.1.63.3 Series.ChangeConfiguration(ChartType, AxisType) . . . 439
1.8.1.63.4 Series.ChartType . . . 439
1.8.1.63.5 Series.DataLabels . . . 439
1.8.1.63.6 Series.DataPointMarker . . . 440
1.8.1.63.7 Series.ErrorBarX . . . 440
1.8.1.63.8 Series.ErrorBarY . . . 440
1.8.1.63.9 Series.HasShadow . . . 440
1.8.1.63.10 Series.HasSmoothedLine . . . 441
1.8.1.63.11 Series.Interior . . . 441
1.8.1.63.12 Series.Item(Int32) . . . 441
1.8.1.63.13 Series.LegendEntryFont . . . 442
1.8.1.63.14 Series.Line . . . 442
1.8.1.63.15 Series.Name . . . 442
1.8.1.63.16 Series.NameFormula . . . 442
1.8.1.63.17 Series.ScatterValues . . . 443
1.8.1.63.18 Series.SettingsBarColumn . . . 443
1.8.1.63.19 Series.SettingsLineAreaScatter . . . 443
1.8.1.63.20 Series.SettingsPieDoughnut . . . 443
1.8.1.63.21 Series.ShowLegendEntry . . . 444
1.8.1.63.22 Series.Trendlines . . . 444
1.8.1.63.23 Series.Values . . . 444
1.8.1.63.24 Series.VaryColorsByPoint . . . 445
1.8.1.64 SeriesCollection . . . 445
1.8.1.64.1 SeriesCollection.CategoryData . . . 446
1.8.1.64.2 SeriesCollection.Count . . . 446
1.8.1.64.3 SeriesCollection.CreateSeries . . . 446
1.8.1.64.4 SeriesCollection.GetEnumerator() . . . 449
1.8.1.64.5 SeriesCollection.Item(Int32) . . . 450
1.8.1.64.6 SeriesCollection.Remove . . . 450
1.8.1.64.7 SeriesCollection.SecondaryCategoryData . . . 451
1.8.1.65 SeriesDataLabel . . . 451
1.8.1.65.1 SeriesDataLabel.ContainsBubbleSize . . . 452
1.8.1.65.2 SeriesDataLabel.ContainsCategoryName . . . 452
1.8.1.65.3 SeriesDataLabel.ContainsSeriesName . . . 452
1.8.1.65.4 SeriesDataLabel.ContainsValue . . . 453
1.8.1.65.5 SeriesDataLabel.ContainsValueAsPercentage . . . 453
1.8.1.65.6 SeriesDataLabel.DataLabelPlacement . . . 453
1.8.1.65.7 SeriesDataLabel.LabelItemSeperator . . . 454
1.8.1.65.8 SeriesDataLabel.ShowLegendKey . . . 454
1.8.1.66 SettingsBarColumn . . . 454
1.8.1.66.1 SettingsBarColumn.DataShapeType . . . 455
1.8.1.66.2 SettingsBarColumn.Overlap . . . 455
1.8.1.66.3 SettingsBarColumn.ShapeType . . . 455
1.8.1.66.4 SettingsBarColumn.SpaceBetweenCategories . . . 456
1.8.1.67 SettingsLineAreaScatter . . . 456
1.8.1.67.1 SettingsLineAreaScatter.DropLine . . . 457
1.8.1.67.2 SettingsLineAreaScatter.HighLowLine . . . 457
1.8.1.67.3 SettingsLineAreaScatter.ShowDropLines . . . 457
1.8.1.67.4 SettingsLineAreaScatter.ShowHighLowLines . . . 457
1.8.1.67.5 SettingsLineAreaScatter.ShowUpDownBars . . . 458
1.8.1.67.6 SettingsLineAreaScatter.UpDownBar . . . 458
1.8.1.68 SettingsPieDoughnut . . . 458
1.8.1.68.1 SettingsPieDoughnut.AngleOfFirstSlice . . . 459
1.8.1.68.2 SettingsPieDoughnut.BopPopGapSize . . . 459
1.8.1.68.3 SettingsPieDoughnut.BopPopSecondChartSize . . . 459
1.8.1.68.4 SettingsPieDoughnut.BopPopSeriesLines . . . 460
1.8.1.68.5 SettingsPieDoughnut.BopPopSplit . . . 460
1.8.1.68.6 SettingsPieDoughnut.BopPopSplitPercentage . . . 460
1.8.1.68.7 SettingsPieDoughnut.BopPopSplitPosition . . . 461
1.8.1.68.8 SettingsPieDoughnut.BopPopSplitType . . . 461
1.8.1.68.9 SettingsPieDoughnut.BopPopSplitValue . . . 461
1.8.1.68.10 SettingsPieDoughnut.LeaderLines . . . 462
1.8.1.68.11 SettingsPieDoughnut.ShowLeaderLines . . . 462
1.8.1.68.12 SettingsPieDoughnut.SizeOfCenterHole . . . 462

1.8.1.68.13 SettingsPieDoughnut.SlicesDistanceFromCenter . . . 462
1.8.1.68.14 SettingsPieDoughnut.UseDefaultSplitValue . . . 463
1.8.1.69 Shape . . . 463
1.8.1.69.1 Shape.ClearRichTextFormatting() . . . 464
1.8.1.69.2 Shape.FillColor . . . 464
1.8.1.69.3 Shape.FillTransparency . . . 464
1.8.1.69.4 Shape.FitToText . . . 465
1.8.1.69.5 Shape.GetCharacters . . . 465
1.8.1.69.6 Shape.Height . . . 466
1.8.1.69.7 Shape.HorizontalAlignment . . . 466
1.8.1.69.8 Shape.LineColor . . . 466
1.8.1.69.9 Shape.LockAspectRatio . . . 467
1.8.1.69.10 Shape.Name . . . 467
1.8.1.69.11 Shape.ObjectPositioning . . . 467
1.8.1.69.12 Shape.Position . . . 467
1.8.1.69.13 Shape.Rotation . . . 468
1.8.1.69.14 Shape.SetCustomFillColor(Int32, Int32, Int32) . . . 468
1.8.1.69.15 Shape.SetCustomLineColor(Int32, Int32, Int32) . . . 468
1.8.1.69.16 Shape.Text . . . 469
1.8.1.69.17 Shape.Type . . . 469
1.8.1.69.18 Shape.VerticalAlignment . . . 469
1.8.1.69.19 Shape.Width . . . 470
1.8.1.70 Shapes . . . 470
1.8.1.70.1 Shapes.AddShape(Shape) . . . 471
1.8.1.70.2 Shapes.Count . . . 471
1.8.1.70.3 Shapes.CreateShape(ShapeType, Anchor) . . . 471
1.8.1.70.4 Shapes.GetEnumerator() . . . 472
1.8.1.70.5 Shapes.Item . . . 472
1.8.1.70.6 Shapes.ListShapes() . . . 473
1.8.1.70.7 Shapes.Remove . . . 474
1.8.1.71 ShapeType . . . 474
1.8.1.72 Style . . . 478
1.8.1.72.1 Style.BackgroundColor . . . 479
1.8.1.72.2 Style.Border . . . 480
1.8.1.72.3 Style.CellLocked . . . 480
1.8.1.72.4 Style.ColumnInsertBehavior . . . 481
1.8.1.72.5 Style.Font . . . 481
1.8.1.72.6 Style.ForegroundColor . . . 482
1.8.1.72.7 Style.HAlign . . . 482
1.8.1.72.8 Style.HideFormulas . . . 483
1.8.1.72.9 Style.HorizontalAlignment . . . 483
1.8.1.72.10 Style.IndentLevel . . . 484
1.8.1.72.11 Style.JustifyDistributed . . . 484
1.8.1.72.12 Style.NumberFormat . . . 485
1.8.1.72.13 Style.Orientation . . . 486
1.8.1.72.14 Style.Pattern . . . 486
1.8.1.72.15 Style.RowInsertBehavior . . . 487
1.8.1.72.16 Style.ShrinkToFit . . . 487
1.8.1.72.17 Style.TextDir . . . 488
1.8.1.72.18 Style.TextDirection . . . 488
1.8.1.72.19 Style.VAlign . . . 489
1.8.1.72.20 Style.VerticalAlignment . . . 489
1.8.1.72.21 Style.WrapText . . . 490
1.8.1.73 Trendline . . . 490
1.8.1.73.1 Trendline.BackcastPeriods . . . 491
1.8.1.73.2 Trendline.ClearIntercept() . . . 491
1.8.1.73.3 Trendline.ForecastPeriods . . . 492
1.8.1.73.4 Trendline.Intercept . . . 492
1.8.1.73.5 Trendline.LegendEntryFont . . . 492
1.8.1.73.6 Trendline.Line . . . 492
1.8.1.73.7 Trendline.Name . . . 493
1.8.1.73.8 Trendline.Regression . . . 493
1.8.1.73.9 Trendline.RegressionType . . . 493
1.8.1.73.10 Trendline.RegressionValue . . . 494
1.8.1.73.11 Trendline.ShowEquation . . . 494
1.8.1.73.12 Trendline.ShowLegendEntry . . . 494
1.8.1.73.13 Trendline.ShowRSquaredValue . . . 494
1.8.1.73.14 Trendline.TrendlineLabel . . . 495
1.8.1.73.15 Trendline.UseIntercept . . . 495
1.8.1.74 Trendlines . . . 495
1.8.1.74.1 Trendlines.Count . . . 496
1.8.1.74.2 Trendlines.CreateTrendline(Trendline.Regression) . . . 496
1.8.1.74.3 Trendlines.GetEnumerator() . . . 497
1.8.1.74.4 Trendlines.Item(Int32) . . . 497
1.8.1.74.5 Trendlines.Remove . . . 497

1.8.1.75 UpDownBar . . . 498
1.8.1.75.1 UpDownBar.DownBarBorderLine . . . 499
1.8.1.75.2 UpDownBar.DownBarInterior . . . 499
1.8.1.75.3 UpDownBar.GapBetweenBars . . . 499
1.8.1.75.4 UpDownBar.UpBarBorderLine . . . 499
1.8.1.75.5 UpDownBar.UpBarInterior . . . 500
1.8.1.76 ValueAxis . . . 500
1.8.1.76.1 ValueAxis.CategoryAxisCrossMaxValue . . . 501
1.8.1.76.2 ValueAxis.CategoryCrossingValue . . . 501
1.8.1.76.3 ValueAxis.MajorIncrement . . . 501
1.8.1.76.4 ValueAxis.MaxValue . . . 501
1.8.1.76.5 ValueAxis.MinorIncrement . . . 502
1.8.1.76.6 ValueAxis.MinValue . . . 502
1.8.1.76.7 ValueAxis.UseLogarithmicScale . . . 502
1.8.1.77 Workbook . . . 502
1.8.1.77.1 Workbook.CreateConditionalFormat() . . . 505
1.8.1.77.2 Workbook.CreateCopyPasteProperties(CopyPasteProperties.CopyPasteType) . . . 505
1.8.1.77.3 Workbook.CreateDataImportProperties() . . . 506
1.8.1.77.4 Workbook.CreateDataValidation . . . 507
1.8.1.77.5 Workbook.CreateNamedRange . . . 512
1.8.1.77.6 Workbook.CreateNamedStyle(String) . . . 513
1.8.1.77.7 Workbook.CreateRange . . . 514
1.8.1.77.8 Workbook.CreateStyle() . . . 515
1.8.1.77.9 Workbook.DisplayTabs . . . 516
1.8.1.77.10 Workbook.DocumentProperties . . . 516
1.8.1.77.11 Workbook.EncryptPassword . . . 517
1.8.1.77.12 Workbook.ExcludeMacros . . . 517
1.8.1.77.13 Workbook.FirstDisplayedTab . . . 518
1.8.1.77.14 Workbook.GetEnumerator() . . . 519
1.8.1.77.15 Workbook.GetNamedObject(String) . . . 519
1.8.1.77.16 Workbook.GetNamedRange(String) . . . 520
1.8.1.77.17 Workbook.GetNamedStyle(String) . . . 520
1.8.1.77.18 Workbook.GetTextHeight(String, Font, Double) . . . 521
1.8.1.77.19 Workbook.GetTextWidth(String, Font, Double) . . . 522
1.8.1.77.20 Workbook.Height . . . 523
1.8.1.77.21 Workbook.HorizontalPosition . . . 524
1.8.1.77.22 Workbook.IsProtected . . . 524
1.8.1.77.23 Workbook.Item . . . 525
1.8.1.77.24 Workbook.NamedRanges . . . 526
1.8.1.77.25 Workbook.NumberFormat . . . 527
1.8.1.77.26 Workbook.Palette . . . 527
1.8.1.77.27 Workbook.PasswordToModify . . . 528
1.8.1.77.28 Workbook.Protect(String) . . . 528
1.8.1.77.29 Workbook.ProtectPasswordHash . . . 529
1.8.1.77.30 Workbook.ReadOnly . . . 529
1.8.1.77.31 Workbook.ReadOnlyRecommended . . . 530
1.8.1.77.32 Workbook.ShowHorizontalScrollBar . . . 530
1.8.1.77.33 Workbook.ShowVerticalScrollBar . . . 531
1.8.1.77.34 Workbook.TabRatio . . . 531
1.8.1.77.35 Workbook.Unprotect() . . . 532
1.8.1.77.36 Workbook.UseRCFormulaNotation . . . 532
1.8.1.77.37 Workbook.VerticalPosition . . . 533
1.8.1.77.38 Workbook.Width . . . 534
1.8.1.77.39 Workbook.Worksheets . . . 534
1.8.1.78 Worksheet . . . 535
1.8.1.78.1 Worksheet.Cells . . . 539
1.8.1.78.2 Worksheet.Charts . . . 540
1.8.1.78.3 Worksheet.Comments . . . 540
1.8.1.78.4 Worksheet.Constants . . . 541
1.8.1.78.5 Worksheet.CopyPaste . . . 541
1.8.1.78.6 Worksheet.CreateAnchor(Int32, Int32, Double, Double) . . . 549
1.8.1.78.7 Worksheet.CreateArea . . . 550
1.8.1.78.8 Worksheet.CreateAreaOfColumns(Int32, Int32) . . . 552
1.8.1.78.9 Worksheet.CreateAreaOfRows(Int32, Int32) . . . 552
1.8.1.78.10 Worksheet.CreateNamedRange . . . 553
1.8.1.78.11 Worksheet.CreateRange(String) . . . 555
1.8.1.78.12 Worksheet.DeleteColumn(Int32) . . . 555
1.8.1.78.13 Worksheet.DeleteColumns(Int32, Int32) . . . 556
1.8.1.78.14 Worksheet.DeleteRow(Int32) . . . 556
1.8.1.78.15 Worksheet.DeleteRows(Int32, Int32) . . . 557
1.8.1.78.16 Worksheet.FirstShownColumn . . . 557
1.8.1.78.17 Worksheet.FirstShownRow . . . 558
1.8.1.78.18 Worksheet.FreezePanes . . . 559
1.8.1.78.19 Worksheet.GetColumnProperties(Int32) . . . 559
1.8.1.78.20 Worksheet.GetNamedObject(String) . . . 560

1.8.1.78.21 Worksheet.GetNamedRange(String) . . . 560
1.8.1.78.22 Worksheet.GetRowProperties(Int32) . . . 561
1.8.1.78.23 Worksheet.GridlinesColor . . . 561
1.8.1.78.24 Worksheet.GroupColumns(Int32, Int32, Boolean) . . . 562
1.8.1.78.25 Worksheet.GroupRows(Int32, Int32, Boolean) . . . 563
1.8.1.78.26 Worksheet.Hyperlinks . . . 563
1.8.1.78.27 Worksheet.ImportData . . . 564
1.8.1.78.28 Worksheet.InsertColumn . . . 572
1.8.1.78.29 Worksheet.InsertColumns . . . 574
1.8.1.78.30 Worksheet.InsertHorizontalPageBreak(Cell) . . . 576
1.8.1.78.31 Worksheet.InsertRow . . . 577
1.8.1.78.32 Worksheet.InsertRows . . . 579
1.8.1.78.33 Worksheet.InsertVerticalPageBreak(Cell) . . . 581
1.8.1.78.34 Worksheet.IsProtected . . . 581
1.8.1.78.35 Worksheet.IsSelected . . . 582
1.8.1.78.36 Worksheet.Item . . . 582
1.8.1.78.37 Worksheet.Name . . . 584
1.8.1.78.38 Worksheet.NamedRanges . . . 584
1.8.1.78.39 Worksheet.PageSetup . . . 585
1.8.1.78.40 Worksheet.Pictures . . . 585
1.8.1.78.41 Worksheet.PopulatedCells . . . 585
1.8.1.78.42 Worksheet.Position . . . 586
1.8.1.78.43 Worksheet.Protect(String) . . . 586
1.8.1.78.44 Worksheet.ProtectPasswordHash . . . 587
1.8.1.78.45 Worksheet.Select() . . . 587
1.8.1.78.46 Worksheet.ShapeGroups . . . 588
1.8.1.78.47 Worksheet.Shapes . . . 588
1.8.1.78.48 Worksheet.SheetViewState . . . 589
1.8.1.78.49 Worksheet.SheetVisibility . . . 589
1.8.1.78.50 Worksheet.ShowFormulas . . . 589
1.8.1.78.51 Worksheet.ShowGridlines . . . 590
1.8.1.78.52 Worksheet.ShowRowColHeaders . . . 591
1.8.1.78.53 Worksheet.ShowZeroValues . . . 591
1.8.1.78.54 Worksheet.StandardHeight . . . 592
1.8.1.78.55 Worksheet.StandardWidth . . . 592
1.8.1.78.56 Worksheet.StandardWidthInChars . . . 593
1.8.1.78.57 Worksheet.SummaryColumns . . . 593
1.8.1.78.58 Worksheet.SummaryColumnsLocation . . . 594
1.8.1.78.59 Worksheet.SummaryRows . . . 594
1.8.1.78.60 Worksheet.SummaryRowsLocation . . . 595
1.8.1.78.61 Worksheet.TabColor . . . 595
1.8.1.78.62 Worksheet.UngroupColumns(Int32, Int32) . . . 596
1.8.1.78.63 Worksheet.UngroupRows(Int32, Int32) . . . 596
1.8.1.78.64 Worksheet.Unprotect() . . . 597
1.8.1.78.65 Worksheet.ViewState . . . 597
1.8.1.78.66 Worksheet.Visibility . . . 598
1.8.1.78.67 Worksheet.Workbook . . . 598
1.8.1.78.68 Worksheet.ZoomPercentage . . . 599
1.8.1.79 Worksheets . . . 599
1.8.1.79.1 Worksheets.CopySheet(Worksheet, Int32, String) . . . 600
1.8.1.79.2 Worksheets.Count . . . 601
1.8.1.79.3 Worksheets.CreateChartsheet . . . 602
1.8.1.79.4 Worksheets.CreateWorksheet . . . 603
1.8.1.79.5 Worksheets.Delete . . . 604
1.8.1.79.6 Worksheets.GetEnumerator() . . . 605
1.8.1.79.7 Worksheets.InsertSheet(String, Int32) . . . 606
1.8.1.79.8 Worksheets.Item . . . 607
1.8.1.79.9 Worksheets.MoveSheet(Worksheet, Int32) . . . 608
1.8.1.79.10 Worksheets.Select(Object()) . . . 608
1.8.2 ExcelTemplate Programmer's Reference . . . 609
1.8.2.1 DataBindingProperties . . . 609
1.8.2.1.1 DataBindingProperties.MaxRows . . . 610
1.8.2.1.2 DataBindingProperties.Transpose . . . 610
1.8.2.1.3 DataBindingProperties.WorksheetName . . . 611
1.8.2.2 ExcelTemplate . . . 612
1.8.2.2.1 ExcelTemplate.ALL_ROWS . . . 615
1.8.2.2.2 ExcelTemplate.BindCellData(Object, String, DataBindingProperties) . . . 615
1.8.2.2.3 ExcelTemplate.BindColumnData . . . 617
1.8.2.2.4 ExcelTemplate.BindData . . . 620
1.8.2.2.5 ExcelTemplate.BindRowData . . . 628
1.8.2.2.6 ExcelTemplate.ContentType . . . 636
1.8.2.2.7 ExcelTemplate.CreateDataBindingProperties() . . . 637
1.8.2.2.8 ExcelTemplate.DecryptPassword . . . 637
1.8.2.2.9 ExcelTemplate.EncryptPassword . . . 638
1.8.2.2.10 ExcelTemplate.ExcelTemplate() . . . 639

1.8.2.2.11 ExcelTemplate.ExcludeMacros . . . 639
1.8.2.2.12 ExcelTemplate.LicenseKey . . . 640
1.8.2.2.13 ExcelTemplate.Open . . . 640
1.8.2.2.14 ExcelTemplate.PreserveStrings . . . 643
1.8.2.2.15 ExcelTemplate.Process() . . . 644
1.8.2.2.16 ExcelTemplate.RemoveExtraDataMarkers . . . 645
1.8.2.2.17 ExcelTemplate.Save . . . 646
1.8.2.2.18 ExcelTemplate.SetCellDataSource(Object, String) . . . 651
1.8.2.2.19 ExcelTemplate.SetColumnDataSource . . . 652
1.8.2.2.20 ExcelTemplate.SetDataSource . . . 658
1.8.2.2.21 ExcelTemplate.SetRowDataSource . . . 673
1.8.2.2.22 ExcelTemplate.StretchCellReferencesInCharts . . . 680
1.8.2.2.23 ExcelTemplate.Version . . . 681
1.8.3 SharePointIntegration Programmer's Reference . . . 682
1.8.3.1 SharePointIntegration . . . 682
1.8.3.1.1 SharePointIntegration.BindData(ExcelTemplate, Microsoft.SharePoint.SPList, String, DataBindingProperties) 683
1.8.3.1.2 SharePointIntegration.BindData(ExcelTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties) . . . 684
1.8.3.1.3 SharePointIntegration.BindRowData(ExcelTemplate, Microsoft.SharePoint.SPList, String, DataBindingProperties) 685
1.8.3.1.4 SharePointIntegration.BindRowData(ExcelTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties) . . . 686
1.8.3.1.5 SharePointIntegration.ImportData(Area, Microsoft.SharePoint.SPList, DataImportProperties) . . . 687
1.8.3.1.6 SharePointIntegration.ImportData(Area, Microsoft.SharePoint.SPList) . . . 688
1.8.3.1.7 SharePointIntegration.ImportData(Area, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties) . . . 689
1.8.3.1.8 SharePointIntegration.ImportData(Area, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList) . . . 690
1.8.3.1.9 SharePointIntegration.ImportData(Worksheet, Cell, Microsoft.SharePoint.SPList, DataImportProperties) . . . 691
1.8.3.1.10 SharePointIntegration.ImportData(Worksheet, Cell, Microsoft.SharePoint.SPList) . . . 692
1.8.3.1.11 SharePointIntegration.ImportData(Worksheet, Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties) . . . 692
1.8.3.1.12 SharePointIntegration.ImportData(Worksheet, Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList) 693
1.8.3.1.13 SharePointIntegration.Open(ExcelApplication, Microsoft.SharePoint.SPDocumentLibrary, String) . . . 694
1.8.3.1.14 SharePointIntegration.Open(ExcelApplication, Microsoft.SharePoint.SPListItem, String) . . . 695
1.8.3.1.15 SharePointIntegration.Open(ExcelTemplate, Microsoft.SharePoint.SPDocumentLibrary, String) . . . 696
1.8.3.1.16 SharePointIntegration.Open(ExcelTemplate, Microsoft.SharePoint.SPListItem, String) . . . 697
1.8.3.1.17 SharePointIntegration.Save(ExcelApplication, Workbook, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean) . . . 698
1.8.3.1.18 SharePointIntegration.Save(ExcelApplication, Workbook, Microsoft.SharePoint.SPListItem, String) . . . 699
1.8.3.1.19 SharePointIntegration.Save(ExcelTemplate, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean) . . . 700
1.8.3.1.20 SharePointIntegration.Save(ExcelTemplate, Microsoft.SharePoint.SPListItem, String) . . . 702
1.9 OfficeWriter Links . . . 703
1.10 External Links . . . 704

ExcelWriter for Windows Version 7

- **Introduction**
  describes the basic features and requirements of ExcelWriter.
- **Quick Start**
  allows you to create your first spreadsheet in minutes. This brief tutorial provides a high level overview of creating spreadsheets using ExcelWriter.
- **ExcelTemplate in Depth**
  explains template-driven spreadsheet creation in detail.
- **ExcelApplication in Depth**
  introduces each major area of ExcelWriter's ExcelApplication object, which allows you to programatically open and manipulate an existing spreadsheet or create a new one from scratch.
- **SharePoint Integration in Depth**
  introduces new extension methods for OfficeWriter objects for use in SharePoint Applications.
- **Installation**
  includes instructions for automatic and manual installation. System Administrators will find useful information here.
- **HotCell Technology**
  shows you how to update a server-side data source from client-side Microsoft Excel.
- **Programmer's Reference**
  describes each object, method, and property.
- **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 ExcelWriter?
- Features and Benefits
- ExcelWriter Editions
- New Features
- Requirements
- Office 2007 Support
- Frequently Asked Questions
- Change Log

What Is ExcelWriter?

SoftArtisans ExcelWriter is a high-performance pure .NET solution that generates native Microsoft Excel spreadsheets on a Web server. A few simple lines of code generate editable presentation-quality spreadsheets that can be saved on the server or viewed instantly by thousands of concurrent users.

The spreadsheets generated by ExcelWriter are in native Microsoft Excel format (.xls) or Office Open XML format (.xlsx) and preserve all of the familiar features associated with Excel workbooks. ExcelWriter offers high-speed performance and, since Microsoft Excel is not required on the Web server running ExcelWriter, your Web applications have licensing flexibility and ease of deployment.

Using ExcelWriter, you can generate a workbook from an existing Excel file or from code:

Using a Template	**Maximum Speed** To generate spreadsheets with the greatest speed: 1. In Microsoft Excel, create a template file containing data markers that will bind to one or more data sources. 2. In code, use ExcelWriter to open the template, set data sources, populate data markers, and save the file, stream it to the client, or both.

Using an Existing Spreadsheet	**Increased Flexibility** When you modify an existing spreadsheet using ExcelWriter code, all Excel features and functionality are preserved in the newly generated spreadsheet. You can easily enhance spreadsheets at runtime with ExcelWriter's extensive design and charting capabilities. In version 7, this option is only available for generating Excel 2003 format (.xls) files.
Pure Coding	**Completely Dynamic** This dynamic approach provides total control, at runtime, of every element in the resulting spreadsheet. In version 7, this option is only available for generating Excel 2003 format (.xls) files.

What ExcelWriter is Not

ExcelWriter is not a visual end-user tool like Microsoft Excel. It is not an ActiveX control that exists on the client. Rather, it user server-side scripting to generate native Excel files that can be read by Microsoft Excel.

ExcelWriter does not use HTML to represent the spreadsheet. Spreadsheets created using ExcelWriter are in the native Microsoft binary format (BIFF8) or the Office Open XML (OOXML) format. This allows you to create rich spreadsheets, including charts and images, and to support of all versions of Excel from Excel 95 through Excel 2010.

ExcelWriter does not have a calculation engine to perform computation on the cells in the spreadsheet. When formulas are included in a spreadsheet, Excel calculates the values when it loads the spreadsheet.

> Since ExcelWriter does not calculate values, evaluation of formulas is not possible during spreadsheet creation.

ExcelWriter does not produce static charts in GIF or JPG/JPEG image format. Rather, it produces fully dynamic 2D or 3D Excel charts capable of referencing data in any worksheet in the workbook that contains the charts.

Features and Benefits

- **Flexible Report Creation:** ExcelWriter generates native Excel spreadsheets from existing Excel binary or Office Open XML files using templates or (in the case of Excel binary files) directly via code.
- **Development Options:** Choice of code-based or template-driven development.
- **Real Excel:** ExcelWriter generates native Excel files including all advanced Excel features including charts, pivot tables, VBA, macros, multiple sheets, image insertion, named ranges, page layouts, and more.
- **High Performance:** Built for the most demanding sites, ExcelWriter easily generates over 100,000 spreadsheets in an hour.
- **Excel Not Required:** Since Excel is not needed on the server, ExcelWriter is perfect for demanding Web sites. See why Microsoft recommends keeping Office off the Server.
- **Delivery Methods:** Save the generated workbook to disk, stream it to a Web browser, or both.
- **Live Data Updates:** HotCell Technology lets you update a server-side data source directly from client-side Microsoft Excel.

ExcelWriter Editions

Production Editions

- **ExcelWriterEE** is for enterprise-class Web applications requiring maximum Excel functionality and high volume of report generation. ExcelWriterEE includes support for both code-based and template-driven development and Reporting Services integration.
- **ExcelWriterSE** supports template-driven development and limited Reporting Services integration. ExcelWriter SE does not include the ExcelApplication object and does not support ExcelTemplate's nesting and grouping functionality.
- **OfficeWriter** unites ExcelWriter and WordWriter, 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, but can generate only one report at a time, which is sufficient for the needs of the development environment. Other restrictions may apply.
- **OfficeWriterSE Developer** offers the same functionality as OfficeWriterSE, but can generate only one report at a time. Other restrictions may apply.

Licensing

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

Compare ExcelWriter Editions

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

New Features

The following is a list of new features added in the most recent major release of ExcelWriter; ExcelWriter v7.5 corresponds to OfficeWriter v4.5. For a list of features added in various minor releases, see OfficeWriter for Excel (ExcelWriter) Version Differences.

New in Version 7.6

- Data markers in Pivot Table cells are no longer processed explicitly (with data being brought in) or implicitly (with blank rows added to the sheet), as described in the new Templates and Pivot Tables page
    - Pivot Table templates no longer require dummy data
    - Pivot Table templates no longer require edit copies set not to refresh on open
    - Shapes and Charts can be placed beside Pivot Tables
- Improved support for files created with Excel 2010
    - ExcelApplication: Excel 2003 (XLS/BIFF8) format files created in Excel 2010 are now supported
    - ExcelTemplate: Excel 2003 (XLS/BIFF8) and Excel 2007 (XLSX/OpenXML) format files created in Excel 2010 are now supported

New in Version 7.5.1

- New **HeaderFooterPicture** object allows you to add pictures to headers and footers
- New **HeaderFooterSection** object gives you finer control over headers and footers
- New **ChartFillPattern** class allows you to use any chart pattern available in Excel 2003
- **SeriesDataLabel.DataLabelPlacement** property
    - Allows you to explicitly set the placement of individual data labels in a series using the DataLabelPlacementType enum

New in Version 7.5

- **SharePoint Integration**
    - ExcelWriter now offers SharePoint Integration extension methods using the new SoftArtisans.OfficeWriter.ExcelWriter.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 spreadsheets as SharePoint List attachments or Document Library Items. For more details see Exporting list as Excel spreadsheet.
- **Cell.SetAddInFormula Method**
    - New Method allows you to add a formula from an add-in (.xla) file

New in Version 7.1

- **Grouping and Nesting**
  - ExcelTemplate now supports grouping and nesting for OOXML (.xlsx, .xlsm, .xltx, .xltm) templates through the use of special markers arranged in grouping blocks. For more details see Grouping and Nesting Markers and Creating a Grouped and Nested Spreadsheet.

New in Version 7.0

- **Office Open XML File Support**
  - ExcelTemplate 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 compressed in a ZIP file. This can result 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. Excel documents in the OOXML format use the .xlsm file extension if they contain embedded macros, and the .xlsx file extension if they do not. For more information, see OfficeWriter for Excel Office 2007 Support.

- **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**
  - ExcelApplication and ExcelTemplate process binary (.xls) files up to 50% faster
  - Files with many formulas are processed up to twice as fast with ExcelTemplate
  - ExcelTemplate uses 40% less memory when importing data into binary files (.xls)
  - ExcelTemplate is 9 times more memory efficient when importing data into OOXML files (.xlsx and .xlsm)
  - ExcelTemplate is 45 times more memory efficient when processing large OOXML files.

- **Iterate over Named Ranges**
  - You can now iterate over all the named ranges in a Worksheet or Workbook

- **Set Fill Transparency for a Shape**
  - The new Shape.FillTransparency property allows you to set the fill transparency of a shape as a real number between 0 (opaque) and 1 (transparent).

- **Bind Data to an Individual Worksheet**
  - Using the new DataBindingProperties object, you can specify a single worksheet on which to bind your data, allowing you to bind different data sets to different worksheets within the same workbook without changing data markers

- **Dedicated .NET DLL**
  - The new SoftArtisans.OfficeWriter.ExcelWriter.dll contains only the pure .NET ExcelApplication and ExcelTemplate classes. The legacy .NET wrapper classes for ExcelWriter COM are no longer included.

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.
- IIS 5 or higher.
- Microsoft .NET Framework Redistributable version 2.0 or later is required.

Server Requirements for Reporting Services Integration

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

- Windows 2000, XP, 2003 Server, 2003 Server R2, Vista, 7, 2008 Server, or 2008 Server R2. 64-bit Windows operating systems are supported.
- Microsoft Reporting Services 2005
- 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 ExcelWriter with Reporting Services 2008, you will need:

- Windows XP, 2003 Server, 2003 Server R2, Vista, 7, 2008 Server, or 2008 Server R2. 64-bit Windows operating systems are supported.
- Microsoft Reporting Services 2008
- 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 Excel is not required on the server. This prevents the performance penalties and instability associated with instantiating multiple copies of the Excel object on a web server.

Client Requirements

No software is required on the client to download and save an Excel Spreadsheet. To open an Excel spreadsheet, you must install one of the following programs on the client:

- Microsoft Excel 95, 97, 2000, 2002 (XP), 2003, or 2007 (see Office 2007 Support).

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

- Microsoft's free Excel Viewer. The Excel Viewer allows the user to view and print an Excel spreadsheet, but not to modify it.

  All formulas will appear as "0" when using the Excel Viewer.

- Another application that supports the Excel format, such as, Quattro Pro or Gnumeric .
- You must download and install the OfficeWriter Assistant cab file (OWAssist.cab) to use HotCell Technology on a client machine. ExcelWriter Enterprise Edition includes OfficeWriter Assistant. OfficeWriter Assistant is not included with ExcelWriterSE.

Client Requirements for Reporting Services Integration

To use ExcelWriter with Reporting Services, you will need:

- Microsoft Windows XP, 2003, 2003 Server R2, 7, 2008 Server, or 2008 Server R2
- Microsoft Excel 2003, 2007, or 2010 (see Office 2007 Support).
- MS Query

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. Open Microsoft Excel.
3. Open the **Data** menu.
4. Select **Import External Data > New Database Query** . You will be prompted to install MS Query.

Office 2007 Support

Microsoft Office 2007 introduced a new XML-based file format, Office Open XML (OOXML) in addition to the binary file format (BIFF 8) used in previous versions. Version 4 and above of OfficeWriter for Excel supports the OOXML format for use with the ExcelTemplate object. (ExcelApplication for OOXML files will be available in a future release) Both the ExcelTemplate and ExcelApplication objects will work fine with Excel binary files (.xls) in Office 2007 and above.

OOXML packages all the text files and embedded binary files that may compose a spreadsheet into a ZIP file. The text files can use any of several XML-based markup languages, such as SpreadsheetML for spreadsheets. The binary files can include formats such BMP for embedded images. Microsoft Office 2007 uses the .xlsm file extension for Excel files containing macros and the .xlsx file extension for Excel files that do not contain macros.

ExcelTemplate supports the new Office 2007 format by opening an ExcelTemplate xlsm or xlsx 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 yet supported in this release.

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

Frequently Asked Questions

Is ExcelWriter a pure .NET solution?

What are ExcelWriter's server requirements?

What ADO.NET data sources can be used with ExcelWriter?

Can ExcelWriter modify an existing spreadsheet?

Can ExcelWriter generate a spreadsheet from a template?

Can I generate a chart using ExcelWriter?

What versions of Microsoft Excel does ExcelWriter support?

Is ExcelWriter a pure .NET solution?

Yes, ExcelWriter is a pure .NET product, written in C#.

What are ExcelWriter's server requirements?

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

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

What ADO.NET data sources can be used with ExcelWriter?

ExcelWriter's ExcelTemplate can import database values to an Excel file from the following ADO.NET objects: DataSet, DataTable, SqlDataReader, OleDbDataReader and AdomdDataReader. The ExcelApplication object can import from from a DataView, DataTable, SqlDataReader, OleDbDataReader, and AdomdDataReader.

Can ExcelWriter modify an existing spreadsheet?

Yes. See Modifying an Existing Spreadsheet.

Can ExcelWriter generate a spreadsheet from a template?

Yes. See How to Use Templates .

> An ExcelWriter template is a file created in Microsoft Excel that contains data markers and can be opened with ExcelTemplate. In the ExcelWriter documentation, the term "template" refers to an ExcelWriter template, not a Microsoft Excel template (.xlt). However, ExcelWriter can open and generate both .xls and .xlt files as well as .xlsx, .xlsm, .xltx, and .xltm files. Use an .xlt file in ExcelWriter the same way you would an .xls file.

Can I generate a chart using ExcelWriter?

Yes. With ExcelApplication, you can create a new chart or modify an existing chart. See Charts.

What versions of Microsoft Excel does ExcelWriter support?

ExcelWriter supports Excel 97 and later versions. ExcelWriter generates BIFF8 format files or Office Open XML format files. BIFF8 is the default file format of all Excel versions starting with Excel 97 until Excel 2007, which uses Office Open XML as its default format.

Change Log

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

Differences between 7.6 and 7.6.1

Features Added

- ExcelApplication is now 20% faster and requires 20% less memory

Issues Resolved

- Specific formula caused "Unreadable Content" error in XLSX file processed by ExcelTemplate in SSRS
- ExcelWriter would return the wrong default character width for Calibri - size 11
- ExcelApplication threw an exception when parsing some SSRS 2008 generated Excel files

- Chart text was not properly set to "Automatic" by default
- Data Series in charts were sometimes not sequenced correctly
- GetHeader and GetFooter threw NullReferenceException when called on newly created worksheets

Differences between 7.5.1 and 7.6

Features Added

- Data markers in Pivot Table cells are no longer processed explicitly (with data being brought in) or implicitly (with blank rows added to the sheet), as described in the new Templates and Pivot Tables page
    - Pivot Table templates no longer require dummy data
    - Pivot Table templates no longer require edit copies set not to refresh on open
    - Shapes and Charts can be placed beside Pivot Tables
- Sparklines now preserved in Excel 2010 created file when processed by ExcelTemplate

Issues Resolved

- ExcelTemplate threw an error on Excel 2010-created files with Excel 2003-format pivot tables
- Shapes such as charts stretched when placed next to pivot tables that were filled using ExcelTemplate
- ExcelWriter generated files no longer open in protected mode in Excel 2010

Differences between 7.5 and 7.5.1

Features Added

- ExcelWriter can now insert images into headers and footers using the new HeaderFooterPicture object.
- The new HeaderFooterSection object allows greater control over headers and footers
- The new ChartFillPattern class provides values for all Excel 2003 patterns
- The new SeriesDataLabel.DataLabelPlacement property allows you to explicitly set the location of the data label on each data point in a Series using the DataLabelPlacementType enum

Issues Resolved

- Issues with PivotCharts in ExcelTemplate
    - PivotCharts missing data in ExcelTemplate reports when streamed to client
    - PivotCharts lose series formatting in OOXML (Excel 2007) file format
- ExcelTemplate did not work with template files created in Excel 2010
- ExcelTemplate did not throw an error if file format limits on the number of characters in a cell were exceeded.
- Processing certain files with comments using ExcelTemplate created a corrupt file
- Calling ExcelTemplate.Process() on files were certain formulas threw an "Unrecognizable symbol" exception.
- ExcelApplication did not throw an error if file format limits on the number of characters in a cell were exceeded.
- Horizontal page breaks could not be inserted on worksheets created using Worksheets.CopySheet
- PageSetup attributes were not applied to worksheets created using Worksheets.CopySheet
- Merging some spreadsheets created using Crystal Reports resulted in a corrupt workbook.
- License Key Manager threw error when being run on a machine with no SoftArtisans license keys in the registry.
- A number of Charting API issues in ExcelApplication
    - Setting Series.Label.ContainsValue broke formatting

    - Setting DataPoint.Interior.Pattern corrupted some files
    - Data Labels set in code not visible in Excel 2003
    - Picture chart fills cause an exception to be thrown when patterns are read
    - ExcelApplication fails to parse Legend visibility correctly
    - Some files were corrupted when a Series was removed from a SeriesCollection
    - Setting a CategoryData formula to a multi-column range threw an invalid formula exception
- ColumnProperties.AutoFitWidth() entered an infinite loop on columns with certain NumberFormat settings
- Cell.Formula threw an exception when attempting to read a formula referencing an external workbook using VLookup
- Some files generated with ExcelTemplate created corrupt files when opened and saved in ExcelApplication

Differences between 7.1 and 7.5

Features Added

- ExcelWriter SharePoint Integration
    - ExcelWriter can now use SharePoint Lists and Views as data sources
    - ExcelWriter can now output spreadsheets to Document Libraries or as List Item attachments
- Cell.SetAddInFormula method

Issues Resolved

- ExcelTemplate can now process files created using Microsoft's OOXML SDK
    - Process method no longer throws exception when processing a worksheet copied using Microsoft's OOXML SDK
    - Process method no longer throws exception when processing a workbook with a large number of worksheets copied using Microsoft's OOXML SDK
- Issues with deleting worksheets in workbooks
    - Deleting a Worksheet in a Workbook with Worksheet-level named ranges no longer corrupts the output spreadsheet
    - Deleting a Worksheet in a Workbook with Workbook-level named ranges no longer throws an "Unable to parse formula" exception when Save is called
    - Calling PageSetup.SetPrintTitleRows or PageSetup.SetPrintTitleColumns on the last sheet of a workbook, then deleting the worksheet no longer throws exception on Save
- Improved Grouping and Nesting Support
    - Headers and Footers are no longer printed when the data source is empty
    - Data marker modifiers now also work on %%value() markers
    - Merged cells are now preserved in spreadsheets with Grouping and Nesting
    - Conditional formatting copied from a header or footer to a data marker row using paste special -> formats is now preserved
    - Shared formulas in headers or footers are changed to equivalent regular formulas, rather than causing a corrupt output spreadsheet
- ExcelTemplate.Process() issues with some templates
    - Process no longer fails on a small number of template files that had previously caused it to fail
    - Process no longer throws a "Cannot find last DBCELL record" after being first opened in ExcelApplication
    - Process no longer throws a "Sheet was not in expected location" error after being first opened in ExcelApplication
- Shape.FillTransparency no longer inverts its value
- Cell values on sheets created using Worksheets.CopySheet no longer reference incorrect values

- Calling ColumnProperties.AutoFitWidth() on a column with a custom NumberFormat no longer causes an Out of Memory exception
- R1C1-style formulas no longer cause an "Unable to parse formula" exception
- Improved ability to handle spreadsheets generated by Crystal Reports
    - ExcelWriter can now open some files generated by Crystal Reports that had previously been impossible to open
    - Small bmp files can now be copied from Excel spreadsheets generated through Crystal Reports

Differences between 7.0 and 7.1

Features Added

- ExcelTemplate can process files created with Microsoft's Office Open XML (OOXML) SDK
- Nesting and grouping formatting enabled for OOXML files. For more information, see Creating Data Markers

Issues Resolved

- Compiler error thrown while using ChartType enumeration in Visual Studio 2008
- Several issues addressed regarding support for files created by the native Excel renderer from SQL Server Reporting Services 2008
    - Accessing worksheets threw object reference exceptions
    - Creating a chart and adding a series corrupted the output file
    - Adding or removing rows or columns, copying worksheets, or creating a named range threw an internal casting exception
    - Creating a range from one or more areas threw a parser exception
    - Accessing or changing colors in the Palette threw Invalid Color Index exception
- Several issues addressed regarding support for files created by the native Excel renderer from SQL Server Reporting Services 2005
    - Missing shared string table from some spreadsheets threw Object Reference exception
    - ColumnProperty.ApplyStyle() failed to get Style interface unless Worksheet.PopulatedCells was first accessed
- ExcelWriter did not process three-letter columns correctly when referencing cells, for example "AAA4"
- AutoFitWidth threw IndexOutOfBounds exception for column with formula formatted as number
- Calling ApplyStyle() or SetStyle() on a ColumnProperties object referencing a column with a formula in it threw an Object Reference exception
- Setting CharacterRun.Font properties for more than 586 cells corrupted the file
- Setting CORREL function with Application object displayed "#VALUE" until user clicked on the cell
- ExcelTemplate.ExcludeMacros property now available when working with OOXML files
- Strings that looked like data markers after a formula is evaluated were treated like data markers
- The wrong content-type header was being sent when streaming OOXML files to the user, causing some browsers to treat them as BIFF8 (.xls) files
- ExcelTemplate no longer inserts a 0 for Null values brought into an OOXML template using a DataReader

Quick Start

- Create a Spreadsheet with ExcelTemplate
- Create a Spreadsheet with ExcelApplication
- Upgrading from ExcelWriter Version 6 to ExcelWriter Version 7

Create a Spreadsheet with ExcelTemplate

ExcelWriter's ExcelTemplate object opens an ExcelWriter template file, populates it with data from a specified data source, and generates a new

Excel spreadsheet. An ExcelWriter **template** is a spreadsheet created in Microsoft Excel that contains data markers.

A **data marker** is a cell value beginning with **%%=** or **%%=$** that specifies a database column, variable, or array to insert in the spreadsheet column containing the marker. **%%=** specifies a database column, a 1-dimensional array that is a data source for a row, or a 2-dimensional array. **%%=$** specifies a variable or a 1-dimensional array that is a data source for a single column.

A data marker may include both a data source name and a column name, for example, **%%=Products.ProductId** where "Products" is a database table and "ProductId" is a column in the table. When the data source is a simple variable or a one-dimensional array, the data marker should begin with **%%=$**. For example, **%%=$RecipientName** where the data source for "RecipientName" is a variable that will be assigned in script.

> The term "template" refers to an ExcelWriter template, not a Microsoft Excel template (.xlt file). However, ExcelWriter can open and generate both .xls and .xlt files as well as .xlsx, .xlsm, and .xltx files. Use an .xlt file in ExcelWriter the same way you would an .xls file.

In the following example, ExcelTemplate opens a template spreadsheet, populates the template's data markers, and generates a new spreadsheet. The template contains two data markers: **%%=$RecipientName** and **%%=$RecipientCompany**. **%%=$** (rather than **%%=**) indicates that the data source is a variable or a single column array. ExcelWriter sets the data sources for the data markers to two simple string variables.

Code sample: Using a String Variable as a Data Source

[C#] | [VB.NET]

Step 1: Create a Template

To generate an Excel spreadsheet with ExcelWriter, first create a template:

1. In Microsoft Excel, create a new file.
2. Place data markers in one or more cells. For example, place the variable data markers **%%=$RecipientName** in cell B11 and **%%=$RecipientCompany** in cell B15.
3. Enter any static cell values in the spreadsheet. Static values and any Excel features included in the template such as formatting, formulas, charts, and macros will be included in the generated spreadsheet.

Step 2: Write the ExcelWriter Code

Next, create an ASP.NET script that uses ExcelTemplate to open the template, populate data markers, and generate a new Excel spreadsheet.

ExcelTemplate is in the SoftArtisans.OfficeWriter.ExcelWriter namespace. The object can be referenced as SoftArtisans.OfficeWriter.ExcelWriter.ExcelTemplate. To minimize typing and errors, use an Import directive to import the namespace to the aspx page and reference the object as ExcelTemplate without the namespace prefix. If you are coding directly in the .aspx page include the following statement after the Page directive:

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

If you are coding in the code-behind page (as in the example above), include a using (C#) or Imports (VB.NET) statement at the top of the code-behind page:

C#

```
using SoftArtisans.OfficeWriter.ExcelWriter;
```

VB.NET

```
Imports SoftArtisans.OfficeWriter.ExcelWriter
```

To generate a new spreadsheet with ExcelWriter:

1. Create an instance of the ExcelTemplate object which represents the template Excel file, for example:

C#

```
ExcelTemplate xlt = new ExcelTemplate();
```

VB.NET

```
Dim xlt As New ExcelTemplate()
```

2. Call ExcelTemplate.Open to open a template Excel file, for example:

C#

```
xlt.Open(@"C:\Templates\Template.xls");
```

VB.NET

```
xlt.Open("C:\Templates\Template.xls")
```

The Open method takes the file path of the template file to open.

3. Use the ExcelTemplate.BindCellData method to bind data sources to the template's data markers, for example:

C#

```
//--- Bind the variables to the data markers
//--- %%=$RecipientName and %%=$RecipientCompany
xlt.BindCellData("J. Smith",
    "RecipientName",
    xlt.CreateDataBindingProperties());
xlt.BindCellData("SoftArtisans",
    "RecipientCompany",
    xlt.CreateDataBindingProperties());
```

VB.NET

```
'--- Bind the variables to the data markers
'--- %%=$RecipientName and %%=$RecipientCompany
xlt.BindCellData("J. Smith", _
    "RecipientName", _
    xlt.CreateDataBindingProperties())
xlt.BindCellData("SoftArtisans", _
    "RecipientCompany", _
    xlt.CreateDataBindingProperties())
```

4. Call ExcelTemplate.Process() to populate the template's data markers with data source values:

C#

```
xlt.Process();
```

VB.NET

```
xlt.Process()
```

The Process method enters data source values in the template's merge fields, and creates the output file (the new spreadsheet) in memory.

5. Call ExcelTemplate.Save to generate a new spreadsheet:

C#

```
xlt.Save(Page.Response, "Output.xls", false);
```

VB.NET

```
xlt.Save(Page.Response, "Output.xls", False)
```

If you pass an HTTP Response object to the Save method, ExcelWriter will stream the generated file to the client. Save's second parameter specifies a name for the generated Excel file; this name will be displayed in the download dialog when the file is streamed to the browser. If the third parameter is set to true and the user chooses to open the file, the file will open in the browser window; if it is set to false (as in the example) and the user chooses to open the file, the file will open in Microsoft Excel.

ExcelWriter allows you to save the generated file on the server or stream it to the client. For more information, see Output Options.

Create a Spreadsheet with ExcelApplication

To create generate an Excel workbook from code - without using an ExcelWriter template - use ExcelApplication, which is the main class generating workbooks using code. ExcelApplication can open, create, and write (save or stream to a browser) workbooks. A single instance of ExcelApplication can generate multiple workbooks.

Step 1: Import ExcelApplication

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

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

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.ExcelWriter;
```

VB.NET

```
Imports SoftArtisans.OfficeWriter.ExcelWriter
```

Step 2: Create an Instance of ExcelApplication

To create an instance of ExcelApplication use:

C#

```
ExcelApplication xla = new ExcelApplication();
```

VB.NET

```
Dim xla As New ExcelApplication()
```

Step 3: Create a Workbook

A Workbook object represents an Excel workbook. To return a Workbook object:

- Call ExcelApplication.Open to open an existing Excel workbook, for example:

C#

```
Workbook wb = xla.Open(@"C:\Reports\Report.xls");
```

VB.NET

```
Dim wb As Workbook = xla.Open("C:\Reports\Report.xls")
```

Or,

- Call ExcelApplication.Create to create a new Excel workbook, for example:

C#

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
```

VB.NET

```
Dim xla As New ExcelApplication()
Dim wb As Workbook = xla.Create()
```

Step 4: Get a Worksheet and Set Cell Values

A Worksheet object represents a single Excel worksheet. To return a Worksheet object use the Workbook.Worksheets property, specifying the sheet by index or name:

C#

```
Worksheet ws = wb.Worksheets[0];
```

VB.NET

```
Dim ws As Worksheet = wb.Worksheets(0)
```

A Cell object represents a single cell in a worksheet. To return a Cell object, use Worksheet.Cells, specifying the cell by 0-based row and column indexes or by Excel-style reference. Use Cell.Value to enter a cell value. The first index of the Cells property is the row index, the second is the column index.

C#

```
ws.Cells[0, 0].Value = "Welcome to SoftArtisans OfficeWriter";
ws.Cells["A3"].Value = "Jan";
ws.Cells["B3"].Value = "Feb";
ws.Cells["C3"].Value = "Mar";
```

VB.NET

```
ws.Cells(0, 0).Value = "Welcome to SoftArtisans OfficeWriter"
ws.Cells("A3").Value = "Jan"
ws.Cells("B3").Value = "Feb"
ws.Cells("C3").Value = "Mar"
```

Step 5: Generate the Workbook

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

C#

```
xla.Save(wb, Page.Response, "myfile.xls", True);
```

VB.NET

```
xla.Save(wb, Page.Response, "myfile.xls", True)
```

Upgrading from ExcelWriter Version 6 to ExcelWriter Version 7

Though ExcelWriter Version 7 is for the most part backwards compatible with applications written for ExcelWriter Version 6, there are a number of changes which must be considered when upgrading from Version 6 to Version 7:

- Installing OfficeWriter Version 4
- Changes in ExcelWriter Properties, Methods, and Enumerations
- New DataBinding Methods

Installing OfficeWriter Version 4

To upgrade ExcelWriter Version 6 to ExcelWriter Version 7, 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 ExcelWriter documentation. Remember to use an OfficeWriter v4 or ExcelWriter v7 license key, as OfficeWriter license keys are version-specific and an OfficeWriter v3/ExcelWriter v6 license key will not register correctly with OfficeWriter v4/ExcelWriter v7.

Changes in ExcelWriter Properties, Methods, and Enumerations

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

The following is a list of methods that have been changed to properties in SoftArtisans ExcelWriter v7. If you use any of these methods in your application, you will need to replace them with the appropriate property call.

ExcelWriter v6 Method	ExcelWriter v7 Property
ChartType.getCategory()	ChartType.Category
ChartType.getName()	ChartType.Name
ChartType.is3D()	ChartType.Is3D
Worksheet.IsSelected()	Worksheet.IsSelected

The following is a list of enumerations for which the getValue method has been replaced with a Value property in ExcelWriter v7. If you used the getValue method of any of these enumerations in your application, you will need to replace them with the appropriate property call.

ExcelWriter v6 Method	ExcelWriter v7 Property
Axis.TickLabelPos.getValue()	Axis.TickLabelPos.Value
Axis.TickType.getValue()	Axis.TickType.Value
AxisLabel.LabelPosition.getValue()	AxisLabel.LabelPosition.Value
Axis.Type.getValue()	Axis.Type.Value
Border.LineStyle.getValue()	Border.LineStyle.Value
Border.Part.getValue()	Border.Part.Value
CategoryAxis.AxisDisplayType.getValue()	CategoryAxis.AxisDisplayType.Value
CategoryAxis.AxisTimeUnit.getValue()	CategoryAxis.AxisTimeUnit.Value
Cell.CellValueType.getValue()	Cell.CellValueType.Value
Chart.ChartLocation.getValue()	Chart.ChartLocation.Value

ChartLine.LineStyle.getValue()	ChartLine.LineStyle.Value
ChartLine.LineWeight.getValue()	ChartLine.LineWeight.Value
ChartPageSetup.PrintedChartSize.getValue()	ChartPageSetup.PrintedChartSize.Value
ChartRegion.HorizontalAlignment.getValue()	ChartRegion.HorizontalAlignment.Value
ChartRegion.VerticalAlignment.getValue()	ChartRegion.VerticalAlignment.Value
ChartType.ChartCategory.getValue()	ChartType.ChartCategory.Value
Condition.Comparison.getValue()	Condition.Comparison.Value
CopyPasteProperties.CopyPasteType.getValue()	CopyPasteProperties.CopyPasteType.Value
DataPointMarker.ShapeType.getValue()	DataPointMarker.ShapeType.Value
DataValidation.ComparisonType.getValue()	DataValidation.ComparisonType.Value
DataValidation .ErrorAlertStyleType.getValue()	DataValidation .ErrorAlertStyleType.Value
DataValidation.ValidationType.getValue()	DataValidation.ValidationType.Value
DocumentProperties.BuiltIn.getValue()	DocumentProperties.BuiltIn.Value
ErrorBar.BarDirection.getValue()	ErrorBar.BarDirection.Value
ErrorBar.ErrorValueType.getValue()	ErrorBar.ErrorValueType.Value
Font.UnderlineStyle.getValue()	Font.UnderlineStyle.Value
Font.VerticalPosition.getValue()	Font.VerticalPosition.Value
Legend.LegendLocation.getValue()	Legend.LegendLocation.Value
NamedObject.BuiltInName.getValue()	NamedObject.BuiltInName.Value
NamedObject.ReturnType.getValue()	NamedObject.ReturnType.Value
NamedObject.Scope.getValue()	NamedObject.Scope.Value
NumberFormat.Color.getValue()	NumberFormat.Color.Value
ObjectPositioning.getValue()	ObjectPositioning.Value
PageSetup.CellErrorsAs.getValue()	PageSetup.CellErrorsAs.Value
PageSetup.PageOrientation.getValue()	PageSetup.PageOrientation.Value
PageSetup.PagePaperSize.getValue()	PageSetup.PagePaperSize.Value
PageSetup.PagePrintOrder.getValue()	PageSetup.PagePrintOrder.Value
Pattern.getValue()	Pattern.Value
SettingsBarColumn.ShapeType.getValue()	SettingsBarColumn.ShapeType.Value
Style.ColumnInsertBehavior.getValue()	Style.ColumnInsertBehavior.Value
Style.HAlign.getValue()	Style.HAlign.Value
Style.RowInsertBehavior.getValue()	Style.RowInsertBehavior.Value
Style.TextDir.getValue()	Style.TextDir.Value
Style.VAlign.getValue()	Style.VAlign.Value
Trendline.Regression.getValue()	Trendline.Regression.Value
Worksheet.SheetViewState.getValue()	Worksheet.SheetViewState.Value
Worksheet.SheetVisibility.getValue()	Worksheet.SheetVisibility.Value
Worksheet.SummaryColumnsLocation.getValue()	Worksheet.SummaryColumnsLocation.Value
Worksheet.SummaryRowsLocation.getValue()	Worksheet.SummaryRowsLocation.Value

The following properties were renamed because of changes in the source code:

ExcelWriter v6 Property	ExcelWriter v7 Equivalent Property
ChartType.Custom.Custom	ChartType.Custom.Combo
ChartType.Scatter.Scatter	ChartType.Scatter.StandardScatter

Additionally, there are a small number of methods which are no longer supported in ExcelWriter v7 and have not been replaced with an alternative method or property:

ChartType.Area.getValue()
ChartType.Bar.getValue()
ChartType.Bubble.getValue()
ChartType.Column.getValue()
ChartType.Custom.getValue()
ChartType.Doughnut.getValue()
ChartType.Line.getValue()
ChartType.Pie.getValue()
ChartType.Radar.getValue()
ChartType.Scatter.getValue()
ChartType.Surface.getValue()
SettingsPieDoughnut.BopPopSplit.getValue()

New DataBinding Methods

The methods used to bind data to a template in ExcelWriter v6 using the ExcelTemplate object have been deprecated in ExcelWriter v7. Though the old SetDataSource methods are still supported for backwards compatibility reasons in ExcelWriter v7, they have been replaced by BindData methods.

The new BindData methods enable the use of the new DataBindingProperties object. Instead of passing a MaxRows or Transpose parameter to BindData as you would SetDataSource, you can set the MaxRows or Transpose properties of the DataBindingProperties object. Additionally, DataBindingProperties includes a WorksheetName property which, when set, will allow you to specify a worksheet if you wish for your data to only be bound to a single worksheet in the template. Multiple BindData calls could bind data to each worksheet in a template from a different data source. Additionally, the same DataBindingProperties object can be used for multiple BindData calls if the same settings are desired for multiple data sets.

This table lists the deprecated SetDataSource methods and their new BindData equivalents:

Deprecated SetDataSource method	New BindData method
SetDataSource	BindData
SetCellDataSource	BindCellData
SetColumnDataSource	BindColumnData
SetRowDataSource	BindRowData

ExcelTemplate in Depth

- How to Use Templates
- Creating Data Markers
- Using an Array as a Data Source
- Using a Database as a Data Source
- Using Charts with ExcelTemplate
- Using Grouping and Nesting with ExcelTemplate
- Advanced Grouping and Nesting
- Creating a Grouped and Nested Spreadsheet

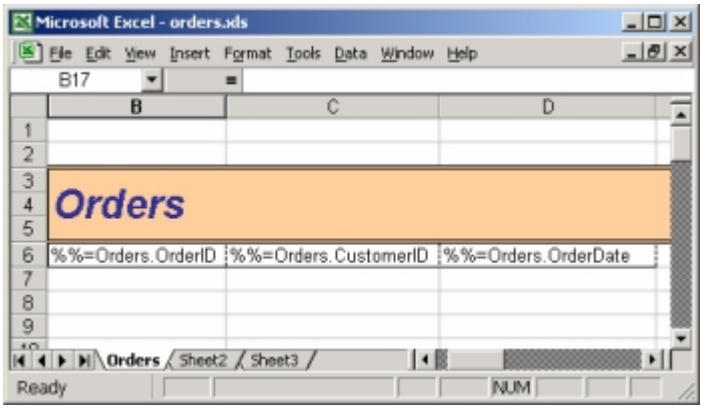
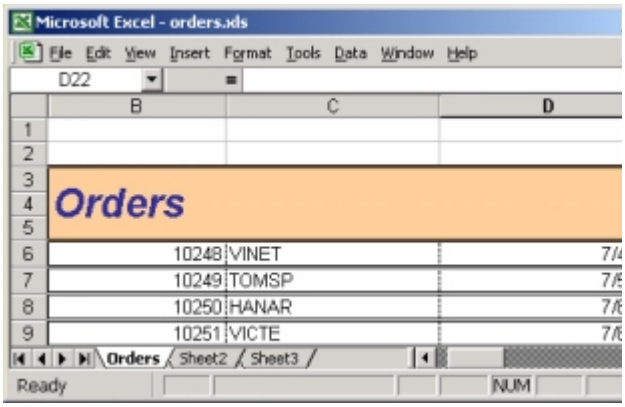
- ExcelTemplate Output Options
- Passing ExcelTemplate to ExcelApplication
- Templates and Pivot Tables
  - Templates and Pivot Tables in Older Versions
- Stretching Formulas
- Drawing Objects and Data Markers

How to Use Templates

An ExcelWriter **template** is an Excel spreadsheet that contains ExcelWriter data markers. A data marker is a cell value beginning with **%%=** or **%%=$** that specifies a database column, variable, or array to insert in the spreadsheet column. A data marker may include modifiers.

Refer to the ExcelTemplate Code Samples to see live demonstrations of various ExcelTemplate applications. Click the **View Template** link on each sample page to see the template.

You can create an ExcelWriter template in Microsoft Excel or in script using the ExcelApplication object. Include data markers where you want to insert values. For example, if cell **B6** contains the data marker **%%=Orders.OrderID**, where Orders represents the Orders table in a database, ExcelWriter will import the **OrderID** column to column **B** in the spreadsheet.

Template	Spreadsheet generated from template
	

A data marker binds in script to a **data source** which may be an array, DataTable, DataSet, or DataReader, and may include modifiers. Data source and field numbers are 1-based. If ExcelWriter encounters **%%=#0[.field]** or **%%=[DataSource.]#0**, an error will occur.

> In ExcelWriter versions before 3.1, data source and field numbers were 0-based. If you upgraded from an earlier version, you may need to modify your code.

Data Marker Formats	Code
Database data marker	`%%=[DataSourceNameOrNumber.]FieldNameOrNumber[(modifier)|(modifier)]`
Variable data marker	`%%=$DataSourceNameOrNumber[(modifier)|(modifier)]`
One-Dimensional Array data marker	`%%=$DataSourceNameOrNumber[(modifier)|(modifier)]`
Two-dimensional Array data marker	`%%=$DataSourceNameOrNumber.ColumnNameOrNumber[(modifier)|(modifier)]`

Example

The following example generates a new spreadsheet from a template that contains two data markers: **%%=$RecipientName** and **%%=$RecipientCompany**. ExcelWriter sets the data sources for the template data markers to two simple string variables.

The ExcelTemplate object represents the template Excel file.

ExcelTemplate is in the SoftArtisans.OfficeWriter.ExcelWriter namespace. The object can be referenced as SoftArtisans.OfficeWriter.ExcelWriter.ExcelTemplate. To minimize typing and errors, use an Import directive to import the namespace to the .aspx page and reference the object as ExcelTemplate without the namespace prefix. If you are coding directly in the .aspx page include the following after the Page directive:

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

If you are coding in the code-behind page, include a using or Imports statement at the top of the code-behind page:

C#

```
using SoftArtisans.OfficeWriter.ExcelWriter;
```

VB.NET

```
Imports SoftArtisans.OfficeWriter.ExcelWriter
```

To generate a new spreadsheet with ExcelWriter:

1. Create an instance of ExcelTemplate, for example:

C#

```
ExcelTemplate xlt = new ExcelTemplate();
```

VB.NET

```
Dim xlt As New ExcelTemplate()
```

1. Call ExcelTemplate.Open() to open a template Excel file, for example:

C#

```
xlt.Open(Application["templatepath"] +
     @"\DataBinding\StringBindingTemplate.xls");
```

VB.NET

```
xlt.Open(Application("templatepath") & _
     "\DataBinding\StringBindingTemplate.xls")
```

The Open method takes the file path and name of the template .xls file to open.

1. Use the ExcelTemplate.BindCellData method to assign data sources to bind to the template's data markers, for example:

C#

```
//--- Bind the variables to the template data markers
//--- %%=$RecipientName and %%=$RecipientCompany
xlt.BindCellData("J. Smith",
     "RecipientName",
     xlt.CreateDataBindingProperties());
xlt.BindCellData("SoftArtisans",
     "RecipientCompany",
     xlt.CreateDataBindingProperties());
```

VB.NET

```
'--- Bind the variables to the template data markers
'--- %%=$RecipientName and %%=$RecipientCompany
xlt.BindCellData("J. Smith",
     "RecipientName",
     xlt.CreateDataBindingProperties())
xlt.BindCellData("SoftArtisans",
     "RecipientCompany",
      xlt.CreateDataBindingProperties())
```

1. Call ExcelTemplate.Process to populate the template's data markers with data source values:

```
xlt.Process();
```

The Process method enters data source values in the template's merge fields, and creates the output file (the new spreadsheet) in memory.

1. Call ExcelTemplate.Save to generate a new spreadsheet:

```
xlt.Save(Page.Response, "StringBinding.xls", false);
```

If you pass Save a Page.Response object, ExcelWriter will stream the generated file to the client. Save's second parameter specifies a name for the generated Excel file; this name will be displayed in the download dialog when the file is streamed to the browser. If the third parameter is set to true and the user chooses to open the file, the file will open in the browser window; if it is set to false (as in the example) and the user chooses to open the file, the file will open in Microsoft Excel.

ExcelWriter allows you to save the generated file on the server or stream it to the client. For more information, see Output Options.

Creating Data Markers

An ExcelWriter template is a spreadsheet created in Microsoft Excel that contains data markers. A **data marker** is a cell value beginning with **%%=** or **%%=$** that specifies a database column, a variable, or an array to insert in the spreadsheet column containing the marker. Data markers are created in Microsoft Excel and bound to data sources in code. When you run the code, ExcelWriter populates the data markers with values from a data source.

Refer to the ExcelTemplate Code Samples to see live demonstrations of various ExcelTemplate applications. Click the "View template" link on each sample page to see the template and view the Data Markers.

A data marker binds in script to a **data source** which may be a variable, an array, or an ADO.NET DataSet, DataTable, or DataReader. A data marker may include modifiers. Data source and field numbers are 1-based. If ExcelWriter encounters **%%=#0[.field]** or **%%=[DataSource.]#0**, an error will occur.

> In ExcelWriter versions before 3.1, data source and field numbers were 0-based. If you upgraded from an earlier version, you may need to modify your code.

Implicitly or explicitly, a data marker always specifies both a **data source** and a **column** - even if the data source is a variable or a one-dimensional array. For example, in the data marker **%%=Products.ProductID**, "Products" is the data source, and "ProductID" is the column. The data marker **%%=ProductID** means the "ProductId" column from the first data source defined in the ExcelWriter 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 ExcelWriter code.
%%=Employee.#3	The third column of the "Employee" data source.
%%=#2.#3	The third column of the second data source defined in the ExcelWriter code.

- Data Marker Names
- Data Marker Formats
- Short Data Markers
- Long Data Markers
- Grouping and Nesting Markers
- Data Marker Modifiers

Data Marker Names

Follow these rules when naming data markers:

- Data source and column names must not include Unicode characters unless using brackets.
- 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 without using brackets: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890_
- Spaces are not allowed anywhere in a data marker except within a column name or data source 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 any Unicode character. Additionally, data source names in the [Data Source] format can include any Unicode character.

Data Marker Formats

Data Marker Formats		
For this data source:	**And this method:**	**Use one of these data marker formats:**
DataSet **DataTable** **DataReader**	BindData SetDataSource(deprecated)	**%%=DataSourceName.ColumnName[(modifier)]** **%%=#DataSourceNumber.#ColumnNumber[(modifier)]** **%%=DataSourceName.#ColumnNumber[(modifier)]** **%%=#DataSourceNumber.ColumnName[(modifier)]** **%%=ColumnName[(modifier)]** **%%=#ColumnNumber[(modifier)]**
Variable	BindCellData SetCellDataSource(deprecated)	**%%=$DataSourceName[(modifier)]** **%%=$#DataSourceNumber[(modifier)]**
1-Dimensional Array	BindColumnData SetColumnDataSource(deprecated)	**%%=$DataSourceName[(modifier)]** **%%=$#DataSourceNumber[(modifier)]**
1-Dimensional Array	BindRowData SetRowDataSource(deprecated)	**%%=DataSourceName.ColumnName[(modifier)]** **%%=#DataSourceNumber.#ColumnNumber[(modifier)]** **%%=DataSourceName.#ColumnNumber[(modifier)]** **%%=#DataSourceNumber.ColumnName[(modifier)]** **%%=ColumnName[(modifier)]** **%%=#ColumnNumber[(modifier)]**
2-dimensional Array	BindData SetDataSource(deprecated)	**%%=DataSourceName.ColumnName[(modifier)]** **%%=#DataSourceNumber.#ColumnNumber[(modifier)]** **%%=DataSourceName.#ColumnNumber[(modifier)]** **%%=#DataSourceNumber.ColumnName[(modifier)]** **%%=ColumnName[(modifier)]** **%%=#ColumnNumber[(modifier)]**

> In ExcelWriter 4, a $ was required for all data markers that bind to 1-dimensional arrays. In ExcelWriter 5 and above, if a 1-dimensional array binds to a row of data markers by the ExcelTemplate.SetRowDataSource or ExcelTemplate.BindRowData methods, the $ should be omitted. However, for backward compatibility, if you include the $, the data marker will be populated without error.

Using Short Data Markers

ExcelWriter supports several full-length and short data marker forms. When you use any of the long data marker 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 ExcelWriter understands a short data marker depends on whether or not the marker contains a $.

ExcelWriter understands **%%=$Employee** as the first column of the data source "Employee." If you omit the $, ExcelWriter will read "Employee" as a column from the first data source defined in the ExcelWriter code. So:

- **%%=$Employee** means the first column in the data source "Employee."
- **%%=Employee** means the "Employee" column in the first data source defined in the ExcelWriter code.

Examples

Form	Example
%%=#ColumnNumber	**%%=#7** Seventh column of the first data source defined in the ExcelWriter code.
%%=$#DataSourceNumber	**%%=$#7** First column of the seventh data source defined in the ExcelWriter code.
%%=ColumnName	**%%=OrderId** "OrderId" column from the first data source defined in the ExcelWriter code.
%%=$DataSourceName	**%%=$Orders** First column in the data source "Orders."

Long Data Markers

When you use any of the long data marker formats, 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 ExcelWriter code, since you must be able to distinguish between data source and column to set the 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 ExcelWriter code.
%%=#DataSourceNumber.#ColumnNumber	**%%=#3.#2** Second column of the third data source defined in the ExcelWriter code.
%%=DataSourceName.[Column Name]	**%%=Employees.[Street Address]** "Street Address" column of the data source "Employees."

Grouping and Nesting Markers

(Enterprise Edition only)

In addition to data markers, ExcelTemplate uses Grouping and Nesting markers to format flat data as it is brought into a spreadsheet. The **%%group**, **%%endgroup**, **%%header**, and **%%footer** markers are used to signify the beginning and ending of the group formatting and the header and footer sections; their rows will not be included in the worksheet. The **%%value(data marker)** marker allows you to include a data marker value in the header or footer of a group, while the **(hide)** modifier, described below, allows you to remove a field from the data marker rows if its values are repeated in the header and footer rows.

> Grouping and Nesting Markers are only evaluated in Office Open XML (.xlsx) files. Grouping and Nesting Markers placed in BIFF8 (.xls) templates will not be evaluated.

Grouping and Nesting Marker Descriptions

(Enterprise Edition only)

%%group	Signifies the beginning of a grouping block. Place this marker in the column of the field by which you wish to group. Grouping and Nesting formatting will be applied to the rows from this marker to the **%%endgroup** marker. This row will not be displayed in the output spreadsheet. The group marker is required for grouping functionality to work.
%%header	Signifies the beginning of the header rows for a particular column in a grouping block. This marker should be placed in a row directly under the row containing the **%%group** markers or other header rows and above the data marker row in the column. All rows between this marker's row and the data marker row, or the next header marker, will be repeated for each new value in the grouping field. The row containing the **%%header** marker will not be displayed in the output spreadsheet. Header markers are optional.
%%value(data marker)	The **%%value(data marker)** marker is used to include the values of a data marker row in the header or footer of a group. The current value of the field whose data marker is placed as a modifier for this marker will be brought into the header or footer, replacing the **%%value()** marker. Value markers are optional.
%%footer	Placed below the data marker rows, this marker signifies the beginning of the footer rows for a particular column in a grouping block. All rows between this marker's row and the **%%endgroup** marker row or the next footer marker will be repeated once for each new value in the grouping field. The row containing the **%%footer** marker will not be displayed in the output spreadsheet. Footer markers are optional.
%%endgroup	Signifies the end of a grouping block. Place this marker in the same column as the **%%group** marker of the group you are ending. The row containing the **%%footer** marker will not be displayed in the output spreadsheet. Only one endgroup marker is required, no matter how many group markers are in the first row of the grouping block.

Example

Below is an example of a grouping block in a template spreadsheet.

%%group				
%%header				
%%value(data.#1)				
%%=data.#1(hide)	**%%=data.#2**	**%%=data.#3**	**%%=data.#4**	**%%=data.#5**
%%footer				
Subtotal row				
%%endgroup				

For the data to be grouped correctly, they must be in a flat format and sorted hierarchically by grouping categories. For the grouping block above, for example, you could use the following data table:

First category	first row	2	4	3
First category	second row	3	4	7
First category	third row	3	5	2
Second category	first row	5	2	9
Second category	second row	4	3	1
Second category	third row	8	5	5

The ExcelTemplate.BindData method should bring the data in as displayed below:

First category				
	first row	2	4	3
	second row	3	4	7
	third row	3	5	2
Subtotal row				
Second category				

	first row	5	2	9
	second row	4	3	1
	third row	8	5	5
Subtotal row				

Data Marker Modifiers

An ExcelWriter data marker can include modifiers, included in parentheses at the end of the data marker, that change the behavior of ExcelTemplate. For example:
%%=Products.ProductID(Lowercase)
If a data marker includes more than one modifier, the modifiers should be separated by commas. Do not include spaces between modifiers.

ExcelWriter supports the following data marker modifiers:

Fieldname	Use the Fieldname modifier to insert database column headings in a spreadsheet. For example, if template cell C5 contains **%%=Products.#2(fieldname)** ExcelWriter will insert the name of column 2 in **C5**.
Uppercase	If a data marker includes the Uppercase modifier, all text values in the data marker column or cell will be displayed in uppercase. If a data marker includes both the Uppercase and Lowercase modifiers, the last will be applied to cell values.
Lowercase	If a data marker includes the Lowercase modifier, all text values in the data marker column or cell will be displayed in lowercase. If a data marker includes both the Uppercase and Lowercase modifiers, the last will be applied to cell values.
Optional	By default, if a data marker's data source name or column name is not bound to a data source in the script, an error will occur. If a data marker contains the Optional modifier, and the data marker is not bound by name to a data source, ExcelWriter will discard the data marker in the generated spreadsheet and will not throw an error. This modifier makes both a data marker's data source and column optional. For example, for this data marker: **%%=Orders.OrderID(Optional)** If either the data source "Orders," or the column "OrderId" does not exist, the data marker will be removed without error. • If a data marker contains a data source number and/or field number (for example **%%=#7.#3**) and is not bound to a data source in script, the data marker will be removed without error whether the "Optional" modifier is present or not. • If the RemoveExtraDataMarkers property is set to true, **all** data markers that do not bind to data sources will be removed without error.
Preserve	The Preserve modifier forces Excel to always insert a string value as a string, instead of attempting to convert the value to a number.
Convert	The Convert modifier forces Excel to always try converting a string to a number. If the conversion fails, the value will be inserted as a string.
Continue	The Continue modifier is used when the user wishes to bind data from a single data source into multiple worksheets. If the data source for the ExcelTemplate uses a forward-only cursor, such as with a DataReader, the second worksheet with the same data marker will automatically start from the row after the last row in the first worksheet. If the data source is scrollable, however, the second worksheet with the same data marker will start from the first row of the data source. The Continue modifier tells the ExcelTemplate object to start from the row after the last row in the previous worksheet when a scrollable data source is being used.
Hide	The Hide modifier removes the field from the spreadsheet after data is bound to the spreadsheet. This is usually used to allow you to remove field values if they are already expressed in a Group header or footer using the **%%value()** marker. Since the data are removed on the server side, they will not be evaluated in any formulas on the spreadsheet.

The (hide) data marker modifier will only be evaluated in Office Open XML (.xlsx) files. (hide) data marker modifiers in BIFF8 (.xls) files will not be evaluated and may throw an exception.

Using an Array as a Data Source

The following sample generates an Excel spreadsheet from a template using arrays as data sources.

Code sample: Using an Array as the Data Source

[C#] | [VB.NET]
The first data marker in the template is **%%=$SimpleArray**. **%%=$** indicates that the data source is either a simple variable or a 1-dimensional array that is a data source for a single column.

> **Data Marker Format for 1-D Arrays**
> For a single column data marker that will be bound to a 1-dimensional array data source by the method BindColumnData, use the data marker format **%%=$DataMarkerName**. For a set of data markers with multiple fields that will be bound to a 1-dimensional array data source by the method BindRowData, use the data marker format **%%=DataMarkerName.Field** (without a **$**). For more information, see Creating Data Markers.

> **Notes on Backward Compatibility**
> In ExcelWriter 7, the SetDataSource, SetCellDataSource, SetColumnDataSource, and SetRowDataSource methods have been deprecated in favor of the new data binding methods BindData, BindCellData, BindColumnData, and BindRowData respectively. In ExcelWriter 4, a **$** was required for all data markers that bind to 1-dimensional arrays. In ExcelWriter 5 and above, if a 1-dimensional array binds to a row of data markers by the method BindRowData and the deprecated method SetRowDataSource, the **$** should be omitted. However, for backward compatibility, if you include the **$**, the data marker will be populated without error.

In ArrayDataSource.aspx.cs, the BindColumnData method is called to set the data source for **%%=$SimpleArray** to a one-dimensional array:

C#

```
string[] onedim = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"};
xlt.BindColumnData(onedim,
    "SimpleArray",
    xlt.CreateDataBindingProperties());
```

VB.NET

```
Dim onedim As String() = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"}
xlt.BindColumnData(onedim, _
    "SimpleArray", _
    xlt.CreateDataBindingProperties())
```

ExcelTemplate.BindColumnData sets a data source for a template column to a 1-dimensional array of objects. The method's first parameter, onedim, is the data source array. The second parameter, "SimpleArray", is the name of the template data marker to which the data source binds. The third parameter, xlt.CreateDataBindingProperties(), is a DataBindingProperties object that contains values that change how the data are bound to the template; in this case, the default property values are used.

The method BindData binds the data markers **%%=TwoDimArray.#1**, **%%=TwoDimArray.#2**, and **%%=TwoDimArray.#3** to a 2-dimensional string array:

C#

```
string[][] twodim = {
    new string[]{"Nancy", "Davolio", "Sales Manager"},
    new string[]{"Michael", "Suyama", "HR Representative"},
    new string[]{"Adrian", "King", "IS Support"}
    };
string[] names = {"FirstName", "LastName", "Position"};
xlt.BindData(twodim,
    names,
    "TwoDimArray",
    xlt.CreateDataBindingProperties());
```

VB.NET

```
Dim twodim()() As String = New String()() { _
    New String(){"Nancy", "Davolio", "Sales Manager"}, _
    New String(){"Michael", "Suyama", "HR Representative"}, _
    New String(){"Adrian", "King", "IS Support"} _
    }
Dim names As String() = {"FirstName", "LastName", "Position"}
xlt.BindData(twodim, _
    names, _
    "TwoDimArray", _
    xlt.CreateDataBindingProperties())
```

BindData's first parameter is the 2-dimensional array of strings to use as the data source. The second parameter specifies an array of column names. The third parameter - "TwoDimArray" - is the name of the template data marker to which the data source binds. The fourth parameter - xlt.CreateDataBindingProperties() - is a DataBindingProperties object that contains values that change how the data are bound to the template; in this case, the default property values are used. There are three "TwoDimArray" data markers; their fields are specified by ordinal (#1, #2, and #3). The columns of values in the data source bind to the data markers by order. That is, the first column of values will populate **%%=TwoDimArray.#1**, the second will populate **%%=TwoDimArray.#2**, and the third **%%=TwoDimArray.#3**.

BindRowData sets a data source for the row of data markers **%%=Address.Street**, **%%=Address.City**, and **%%=Address.State** to a 1-dimensional array:

C#

```
string[] addressvalues = {"3 Brook St.", "Watertown", "MA"};
string[] addressnames = {"Street", "City", "State"};
xlt.BindRowData(addressvalues,
    addressnames,
    "Address",
    xlt.CreateDataBindingProperties());
```

VB.NET

```
Dim addressvalues As String() = {"3 Brook St.", "Watertown", "MA"}
Dim addressnames As String() = {"Street", "City", "State"}
xlt.BindRowData(addressvalues, _
    addressnames, _
    "Address", _
    xlt.CreateDataBindingProperties())
```

The first parameter of BindRowData, addressvalues, specifies the array of values to bind to the data markers. The second parameter, addressnames, is the array of data marker field names. The values in the data source array will bind to the data marker fields by field name. The third parameter, "Address", is the name of the data marker to which ExcelTemplate binds the data. The fourth parameter, xlt.CreateDataBindingProperties(), is a DataBindingProperties object that contains values that change how the data are bound to the template; in this case, the default property values are used.

Data Marker	Code

%%=$SimpleArray	***C#***
	
	`string[] onedim = {"SoftArtisans",`
	`  "OfficeWriter",`
	`  "ExcelTemplate"};`
	`xlt.BindColumnData(onedim,`
	`  "SimpleArray",`
	`  xlt.CreateDataBindingProperties());`
	
	VB.NET
	
	`Dim onedim As String() = {"SoftArtisans", _`
	`  "OfficeWriter", _`
	`  "ExcelTemplate"}`
	`xlt.BindColumnData(onedim, _`
	`  "SimpleArray", _`
	`  xlt.CreateDataBindingProperties())`

%%=TwoDimArray.#1
 	
%%=TwoDimArray.#2
 	
%%=TwoDimArray.#3
	C#
	
	`string[][] twodim = {`
	`  new string[]{"Nancy", "Davolio", "Sales Manager"},`
	`  new string[]{"Michael", "Suyama", "HR Representative"},`
	`  new string[]{"Adrian",`
	`  "King",`
	`  "IS Support"}`
	`  };`
	`string[] names = {"FirstName", "LastName", "Position"};`
	`xlt.BindData(twodim,`
	`  names,`
	`  "TwoDimArray",`
	`  xlt.CreateDataBindingProperties());`
	
	VB.NET
	
	`Dim twodim()() As String = New String()() { _`
	`  New String(){"Nancy", "Davolio", "Sales Manager"}, _`
	`  New String(){"Michael", "Suyama", "HR Representative"}, _`
	`  New String(){"Adrian", _`
	`  "King", _`
	`  "IS Support"}`
	`  }`
	`Dim names As String() = {"FirstName", "LastName", "Position"}`
	`xlt.BindData(twodim, _`
	`  names, _`
	`  "TwoDimArray", _`
	`  xlt.CreateDataBindingProperties())`

%%=Address.Street
%%=Address.City
%%=Address.State
	C#
	
	``
	string[] addressvalues = {"3 Brook St.",
	    "Watertown",
	    "MA"};
	string[] addressnames = {"Street", "City", "State"};
	xlt.BindRowData(addressvalues,
	    addressnames,
	    "Address",
	    xlt.CreateDataBindingProperties());
	``
	
	VB.NET
	
	``
	Dim addressvalues As String() = {"3 Brook St.", _
	    "Watertown", _
	    "MA"}
	Dim addressnames As String() = {"Street", "City", "State"}
	xlt.BindRowData(addressvalues, _
	    addressnames, _
	    "Address", _
	    xlt.CreateDataBindingProperties())
	``

Using a Database as a Data Source

Code Sample: Importing from a Database

[C#] | [VB.NET]
The sample above generates a spreadsheet from the template using ADO.NET objects as the data source. The template contains the following data markers:

Data Markers

`%%=Employee.Name`
`%%=Employee.Title`
`%%=Orders.OrderID`
`%%=Orders.Customer`
`%%=Orders.OrderDate`
`%%=Orders.OrderTotal`

These data markers will be populated in the ASP.NET code by two database data sources. The data source for the %%=Employee.* data markers is an ADO.NET DataTable, and the data source for the %%=Orders.* data markers is an ADO.NET DataSet.

To set a template data source to a DataSet or DataTable using OleDB, import the System.Data and System.Data.OleDB namespaces to the ASP.NET page. For example:

C#

```
using System.Data;
using System.Data.OleDb;
```

VB.NET

```
Imports System.Data
Imports System.Data.OleDb
```

Setting a Data Source to a DataSet with OleDB

To create a DataSet to use as a data source:

1. Create an OleDbConnection object to open a database connection.
2. Create a SQL query to get data from the database.
3. Create an OleDbDataAdapter that will execute the SQL command at the data source and fill the DataSet.
4. Create a DataSet and use the OleDbDataAdapter to fill it with the data retrieved from the database.

For example, in Databasebind.aspx, the following code creates and fills OrdersDs, the DataSet which will be used as the data source for the set of %%=Orders.* data markers.

C#

```
OleDbConnection Conn = new OleDbConnection();
DataSet OrdersDs = null;
try
{
     Conn.ConnectionString = Application["connstring"].ToString();

     //--- SQL Query for orders
     string OrdersSQL = "SELECT Orders.OrderID, " +
          "Customers.CompanyName As Customer, " +
          "Orders.OrderDate, " +
          "([Order Details].UnitPrice * " +
          "[Order Details].Quantity) As [OrderTotal] " +
          "FROM Orders, [Order Details], Customers " +
          "WHERE Orders.OrderID=[Order Details].OrderID AND " +
          "Orders.CustomerID=Customers.CustomerID AND Orders.EmployeeID=?";
     OleDbCommand CmdOrders = new OleDbCommand(OrdersSQL, Conn);
     CmdOrders.Parameters.Add("@EmployeeID", EmployeeId);
     OleDbDataAdapter AdptSales = new OleDbDataAdapter(CmdOrders);
     OrdersDs = new DataSet();
     AdptSales.Fill(OrdersDs, "Orders");
}
```

VB.NET

```
Dim Conn As New OleDbConnection()
Dim OrdersDs As DataSet = Nothing
Try
     Conn.ConnectionString = Application("connstring").ToString()

     '--- SQL Query for orders
     Dim OrdersSQL As String = "SELECT Orders.OrderID, " & _
          "Customers.CompanyName As Customer, " & _
          "Orders.OrderDate, " & _
          "([Order Details].UnitPrice * " & _
          "[Order Details].Quantity) As [OrderTotal] " & _
          "FROM Orders, [Order Details], Customers " & _
          "WHERE Orders.OrderID=[Order Details].OrderID AND " & _
          "Orders.CustomerID=Customers.CustomerID AND Orders.EmployeeID=?"
     Dim CmdOrders As New OleDbCommand(OrdersSQL, Conn)
     CmdOrders.Parameters.Add("@EmployeeID", EmployeeId)
     Dim AdptSales As New OleDbDataAdapter(CmdOrders)
     OrdersDs = New DataSet()
     AdptSales.Fill(OrdersDs, "Orders")
End Try
```

To bind a DataSet to a template data marker, call BindData using the following signature:

```
ExcelTemplate.BindData(DataSet aData,
    String aDataMarkerName,
    DataBindingProperties aDataBindingProperties)
```

This signature takes a DataSet, the name of the data marker to which the data source should bind, and a collection of binding property values. In Databasebind.aspx, this BindData call binds the DataSet OrdersDs to the data markers **%%=Orders.OrderID**, **%%=Orders.Customer**, **%%=Orders.OrderDate**, and **%%=Orders.OrderTotal** in the template:

C#

```
xlt.BindData(OrdersDs,
     "Orders",
     xlt.CreateDataBindingProperties());
```

VB.NET

```
xlt.BindData(OrdersDs, _
     "Orders", _
     xlt.CreateDataBindingProperties())
```

Setting a Data Source to a DataTable with OleDB

To create a DataTable to use as a data source:

1. Create an OleDbConnection object to open a database connection.
2. Create a SQL query to get data from the database.
3. Create an OleDbDataAdapter that will execute the SQL command at the data source and fill the DataTable.
4. Create a DataTable and use the OleDbDataAdapter to fill it with the data retrieved from the database.

For example, in Databasebind.aspx, the following code creates and fills EmployeeDt, the DataTable which will be used as the data source for the set of %%=Employee.* data markers.

C#

```
OleDbConnection Conn = new OleDbConnection();
DataTable EmployeeDt = null;
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);
}
```

VB.NET

```
Dim Conn As New OleDbConnection()
Dim EmployeeDt As DataTable = Nothing
Try
     Conn.ConnectionString = Application("connstring").ToString()

     '--- SQL Query for employee information
     Dim EmployeeSQL As String = "SELECT FirstName & ' ' & LastName As Name, Title " & _
          "FROM Employees WHERE EmployeeID=?"
     Dim CmdEmployee As New OleDbCommand(EmployeeSQL, Conn)
     CmdEmployee.Parameters.Add("@EmployeeID", EmployeeId)
     Dim AdptEmployee As New OleDbDataAdapter(CmdEmployee)
     EmployeeDt = New DataTable()
     AdptEmployee.Fill(EmployeeDt)
End Try
```

To bind a DataTable to a template data marker, call BindData using the following signature:

Signature
`BindData(DataTable aData, String aDataMarkerName, DataBindingProperties aDataBindingProperties)`

This signature takes a DataTable, the name of the data marker to which the data source should bind, and a collection of binding property values. In Databasebind.aspx, this BindData call binds the DataTable EmployeeDt to the data markers **%%=Employee.Name** and **%%=Employee.Title** in the template:

C#

```
xlt.BindData(EmployeeDt,
     "Employee",
     xlt.CreateDataBindingProperties());
```

VB.NET

```
xlt.BindData(EmployeeDt, _
     "Employee", _
     xlt.CreateDataBindingProperties())
```

Using Charts with ExcelTemplate

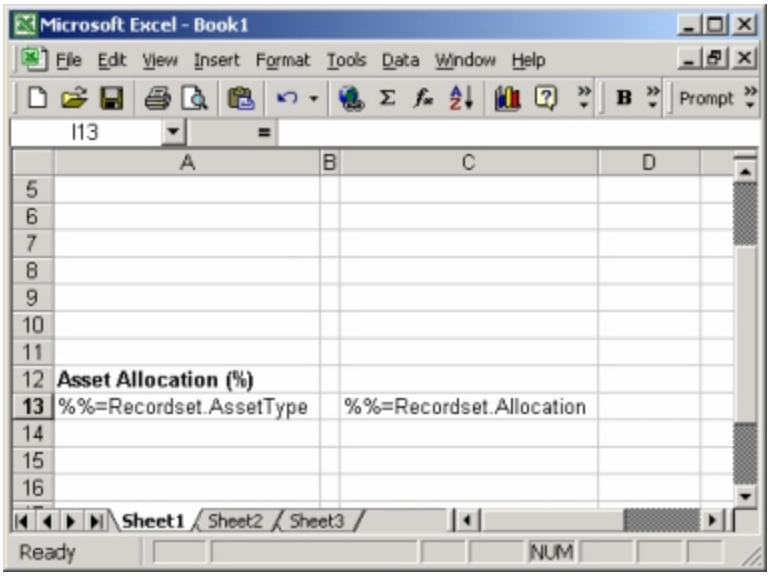
When you include a chart in a template spreadsheet, you can use a data marker column as a data source for the chart. Excel will automatically adjust charts to include the number of rows that ExcelWriter assigns to the data marker column.

Code Sample: Using Charts with ExcelTemplate

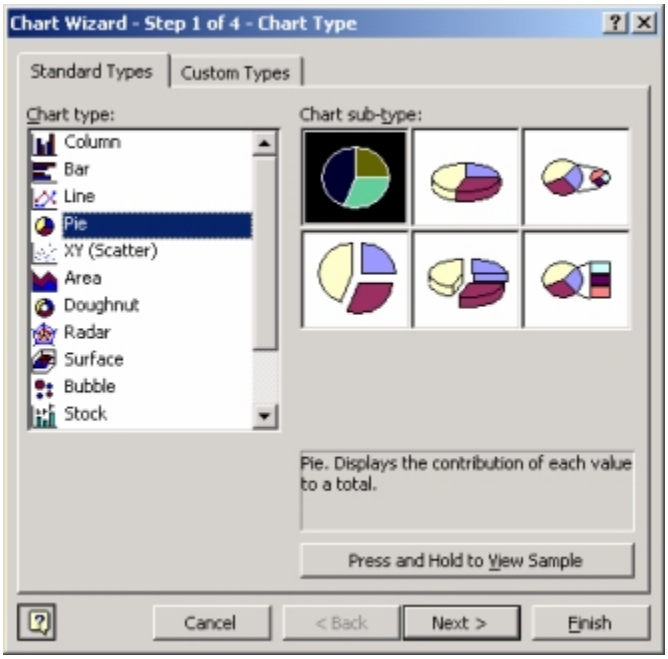
[C#] | [VB.NET]

How to Link a Chart to a Template Data Marker

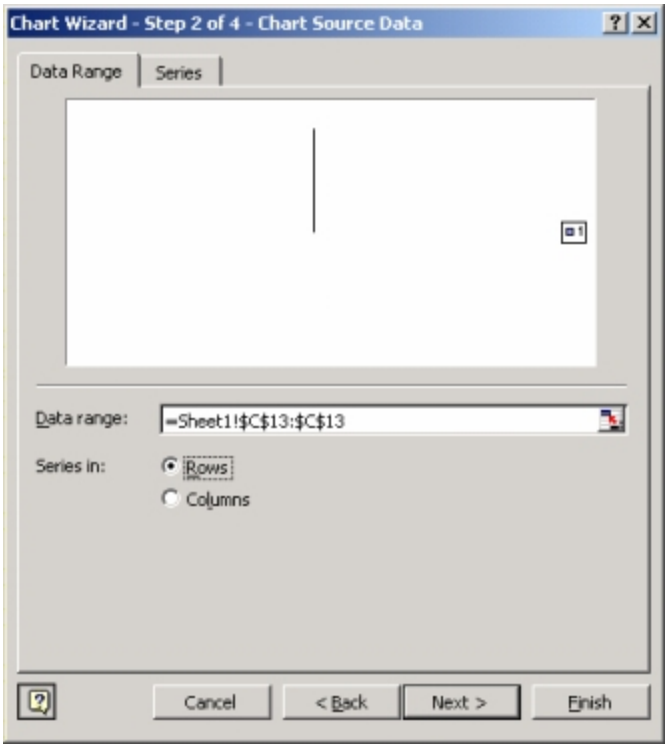
1. In Microsoft Excel, create an ExcelWriter template. For instructions on creating a template, see How to Use Templates.



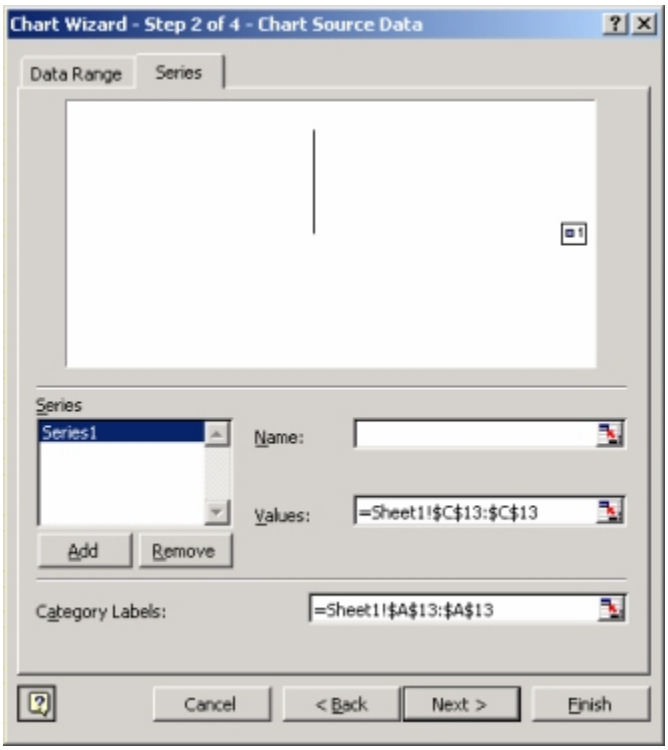
2. Use Excel's Chart Wizard to create a chart in your spreadsheet. Open the **Insert** menu and select **Chart...**, or click the toolbar's Chart Wizard icon. Select the type of chart you would like to create and click **Next**.



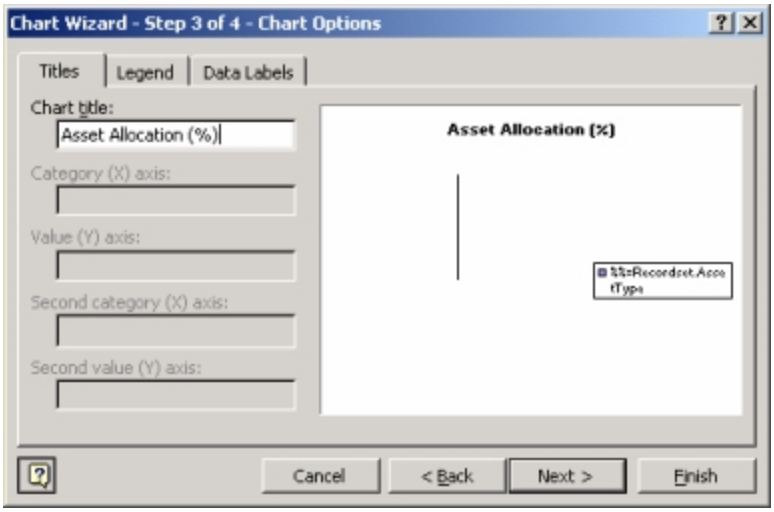
3. In the **Data Range** field, enter the cell that contains the data marker that you want to use as a data source for your chart. Excel requires both a starting point and an ending point in the Data Range Field. So, if your data marker is at cell C13, enter **=Sheet1!$C$13:$C$13**. The number of rows ExcelWriter will insert at the data marker will vary, and Excel will automatically adjust the chart to include all data in the data marker column.



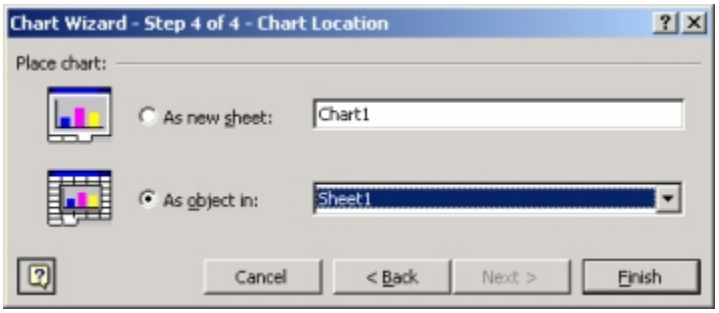
4. Select the **Series** tab, and in the **Category Label** field, enter the range of cells containing the label data marker. Include starting and end points in the category label range. For example, if the label data marker is at cell A13, enter **=Sheet1!$A$13:$A$13**and Excel will automatically adjust the chart for the amount of data that fills the spreadsheet.



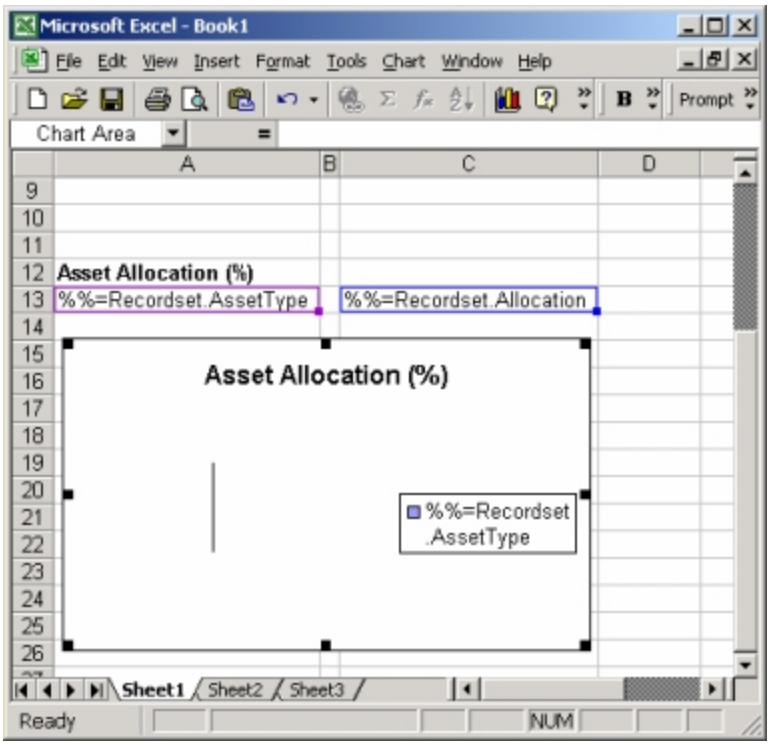
5. Enter a chart title and click **Next**.



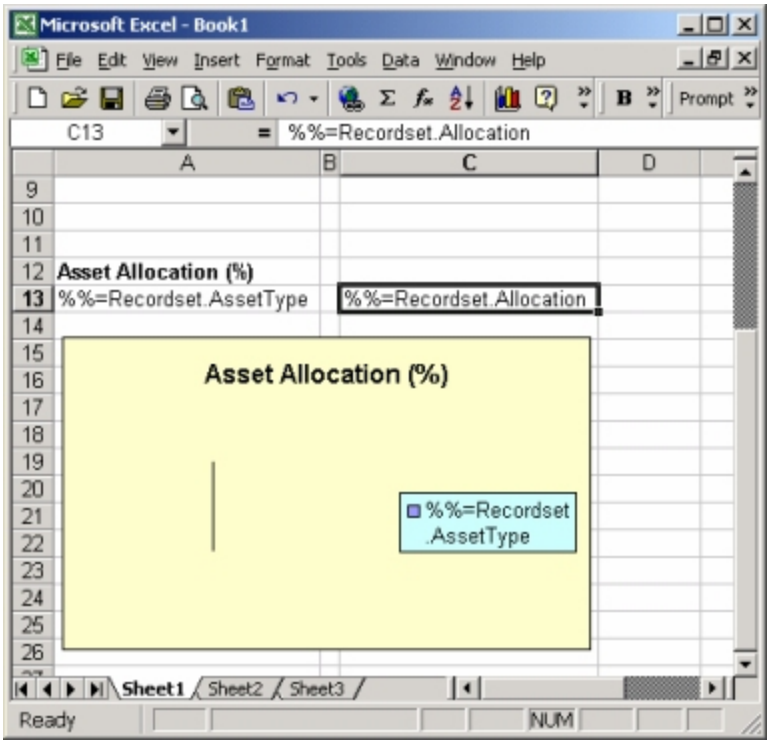
6. Select a chart location and click **Finish** to insert the chart into your spreadsheet.



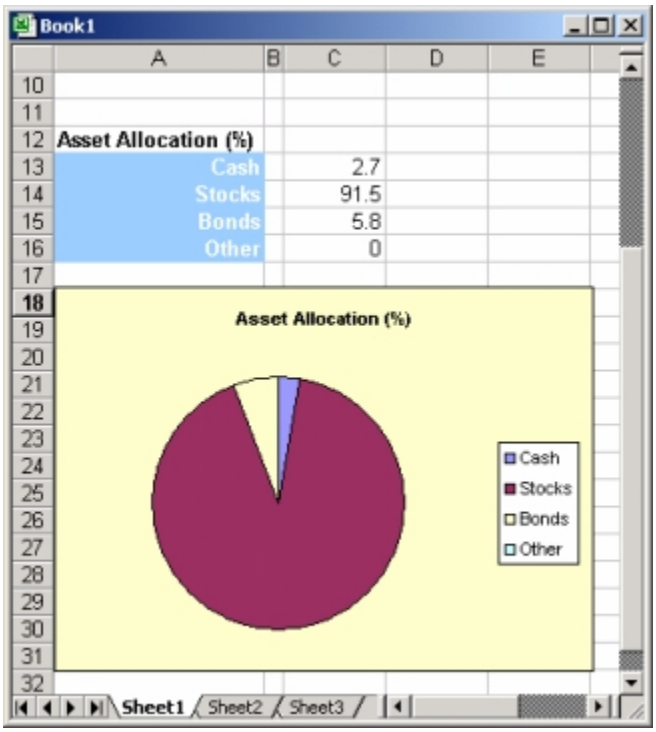
The unpopulated chart is now in your spreadsheet.



7. Format the chart and data marker cells as you would like them to appear when the final spreadsheet is created. Data marker formatting is carried down to each inserted cell when data is dynamically entered into the spreadsheet by ExcelWriter.



8. Run your ExcelWriter application to populate the spreadsheet.



Using Grouping and Nesting with ExcelTemplate

Introduced in build 7.0.0.1261

Starting in version 7.1 of ExcelWriter, you can use grouping and nesting markers to display flat data in a grouped and nested format using the ExcelTemplate object.

Introduction

In order to help organize hierarchical data in a more easily readable format, ExcelTemplate now features grouping and nesting options. The basic feature of grouping and nesting is that ExcelTemplate can remove unnecessary repeated values from a field if it is given hierarchical data in a flat format. This basic functionality is explained in further detail below, while advanced features such as headers and footers are discussed on the Advanced Grouping and Nesting page.

Creating a Data Set for grouping

Though grouping and nesting is meant to show hierarchical data, the input data for a template with grouping and nesting must be in flat format. This means that a data set for nesting and grouping will have repeated values for fields that are higher in the hierarchy. For this tutorial, the data will look like this:

Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty	Line Total
Central	Jillian Carson	Bikes	Mountain Bikes	Black	$2024.99	1	$2024.99
Central	Jillian Carson	Bikes	Mountain Bikes	Black	$2024.99	1	$2024.99
Central	Jillian Carson	Bikes	Mountain Bikes	Silver	$2039.99	1	$2024.99
Central	Jillian Carson	Clothing	Socks	White	$5.70	3	$17.10
Northwest	David Campbell	Accessories	Helmets	Red	$20.19	2	$40.37
Northwest	David Campbell	Accessories	Helmets	Blue	$20.19	1	$20.19
Northwest	David Campbell	Bikes	Mountain Bikes	Black	$2024.99	2	$4049.99
Northwest	David Campbell	Bikes	Mountain Bikes	Black	$2024.99	1	$2024.99

Grouping code

Grouping code is no different from regular ExcelTemplate code. In this tutorial, we use the following ExcelTemplate code:

C#

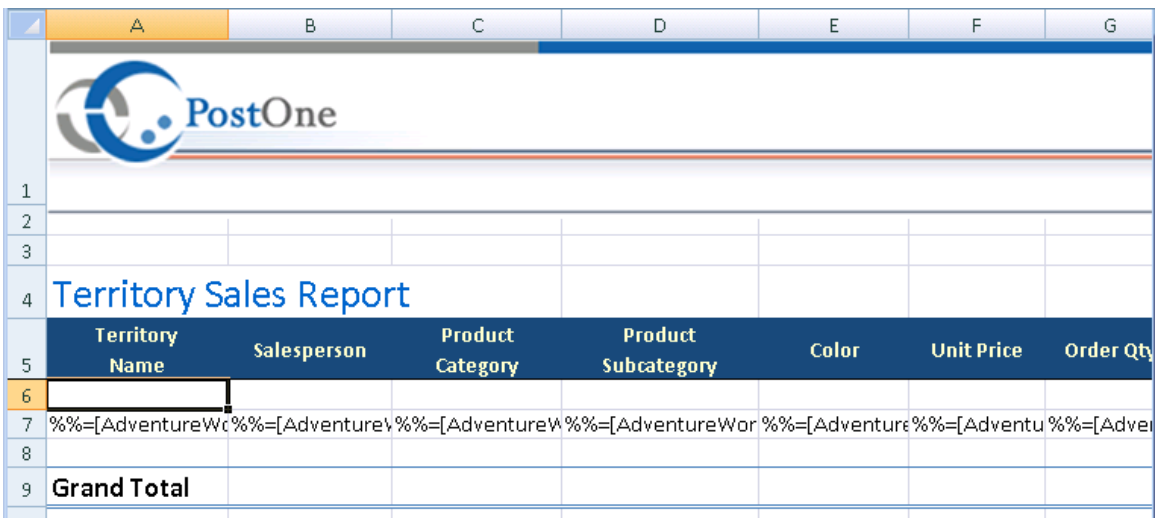
```
DataSet ds = GetData(); //--- Get the Sales data
ExcelTemplate xlt = new ExcelTemplate(); //--- Create a new ExcelTemplate object
xlt.Open(Page.MapPath(@"templates\GroupingTemplate.xlsx")); //--- Open the template

//--- Create the DataBindingProperties object
DataBindingProperties props = xlt.CreateDataBindingProperties();

//--- Bind the data to the template
xlt.BindData(ds, "AdventureWorks", props);

//--- Process the template to import the data
xlt.Process();

//--- Stream the output back to the user
xlt.Save(Page.Response, "Grouped Sales Report.xlsx", false);
```

VB.NET

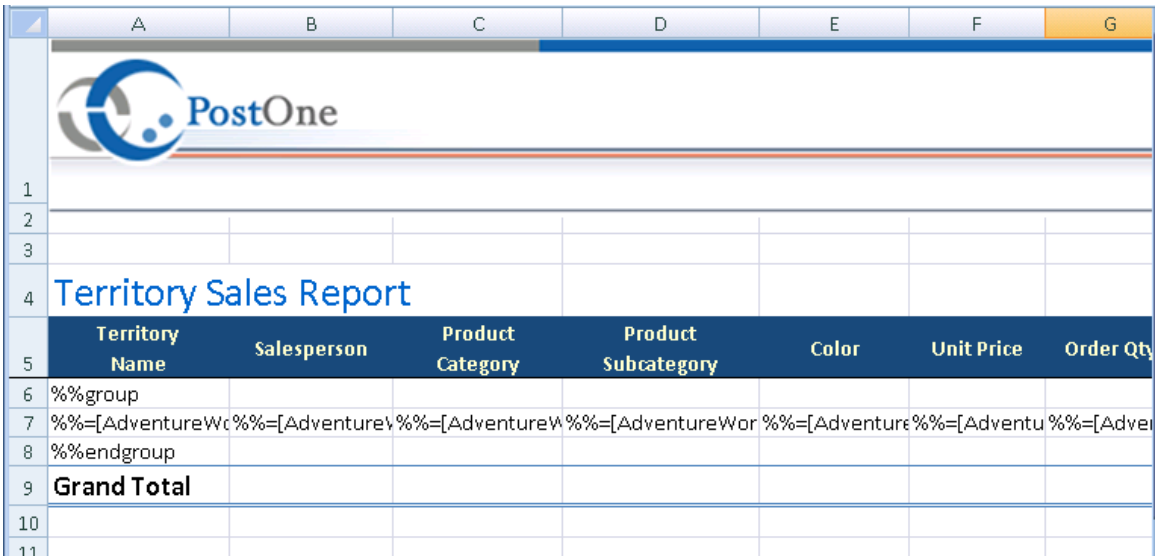
```
Dim ds As DataSet = GetData() '--- Get the Sales data
Dim xlt As New ExcelTemplate() '--- Create a new ExcelTemplate object
xlt.Open(Page.MapPath("templates\GroupingTemplate.xlsx")) '--- Open the template

'--- Create the DataBindingProperties object
Dim props As DataBindingProperties = xlt.CreateDataBindingProperties()

'--- Bind the data to the template
xlt.BindData(ds, "AdventureWorks", props)

'--- Process the template to import the data
xlt.Process()

'--- Stream the output back to the user
xlt.Save(Page.Response, "Grouped Sales Report.xlsx", False)
```

Grouping by a single group

The first element needed for a grouping template is a data row that contains all of the data markers for your data set, as shown below:

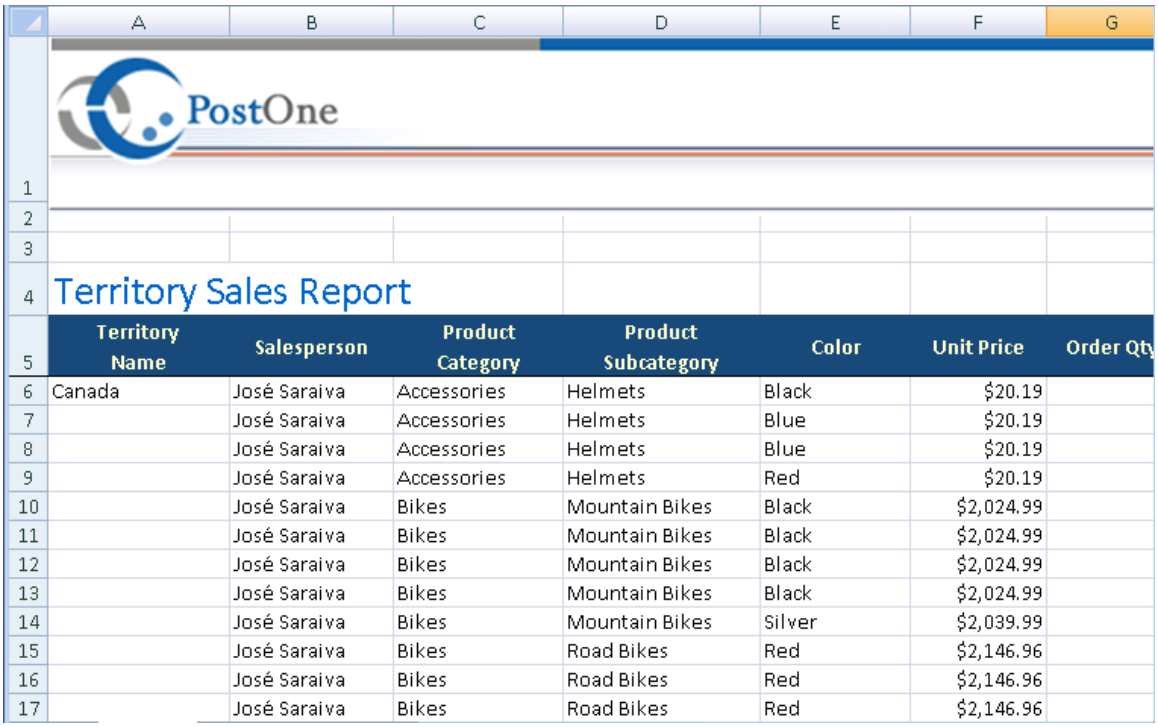


If you bound the data above to this template, it would display exactly as you see it above – in flat form. You can group based on the first column – Territory Name – by adding a **%%group** tag to the cell above the data marker for that column. If you add any **%%group** tag, you must also add an **%%endgroup** tag below the data markers in order to mark the end of the grouping section:



	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
6	%%group						
7	%%=[AdventureWo	%%=[AdventureV	%%=[AdventureW	%%=[AdventureWor	%%=[Adventure	%%=[Adventu	%%=[Adve
8	%%endgroup						
9	**Grand Total**						
10							

The resulting spreadsheet should look like the one below, with all of the extra repeated values from the Territory Name field removed:

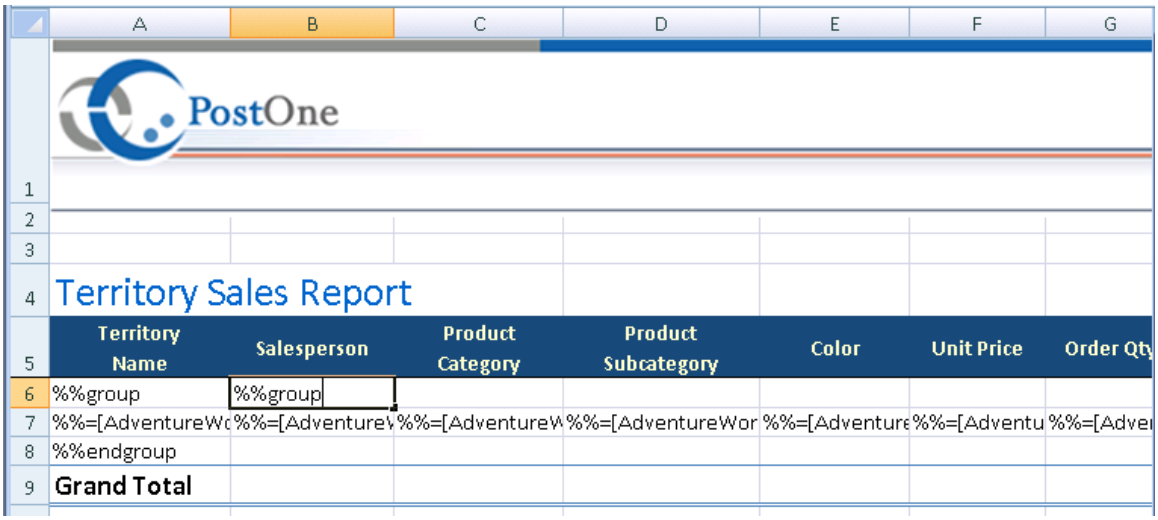


	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
6	Canada	José Saraiva	Accessories	Helmets	Black	$20.19	
7		José Saraiva	Accessories	Helmets	Blue	$20.19	
8		José Saraiva	Accessories	Helmets	Blue	$20.19	
9		José Saraiva	Accessories	Helmets	Red	$20.19	
10		José Saraiva	Bikes	Mountain Bikes	Black	$2,024.99	
11		José Saraiva	Bikes	Mountain Bikes	Black	$2,024.99	
12		José Saraiva	Bikes	Mountain Bikes	Black	$2,024.99	
13		José Saraiva	Bikes	Mountain Bikes	Black	$2,024.99	
14		José Saraiva	Bikes	Mountain Bikes	Silver	$2,039.99	
15		José Saraiva	Bikes	Road Bikes	Red	$2,146.96	
16		José Saraiva	Bikes	Road Bikes	Red	$2,146.96	
17		José Saraiva	Bikes	Road Bikes	Red	$2,146.96	

Notice that the rows with the **%%group** and **%%endgroup** tags are removed as well. Be sure not to place any content in these rows, as it will be automatically removed when data is bound to the template.

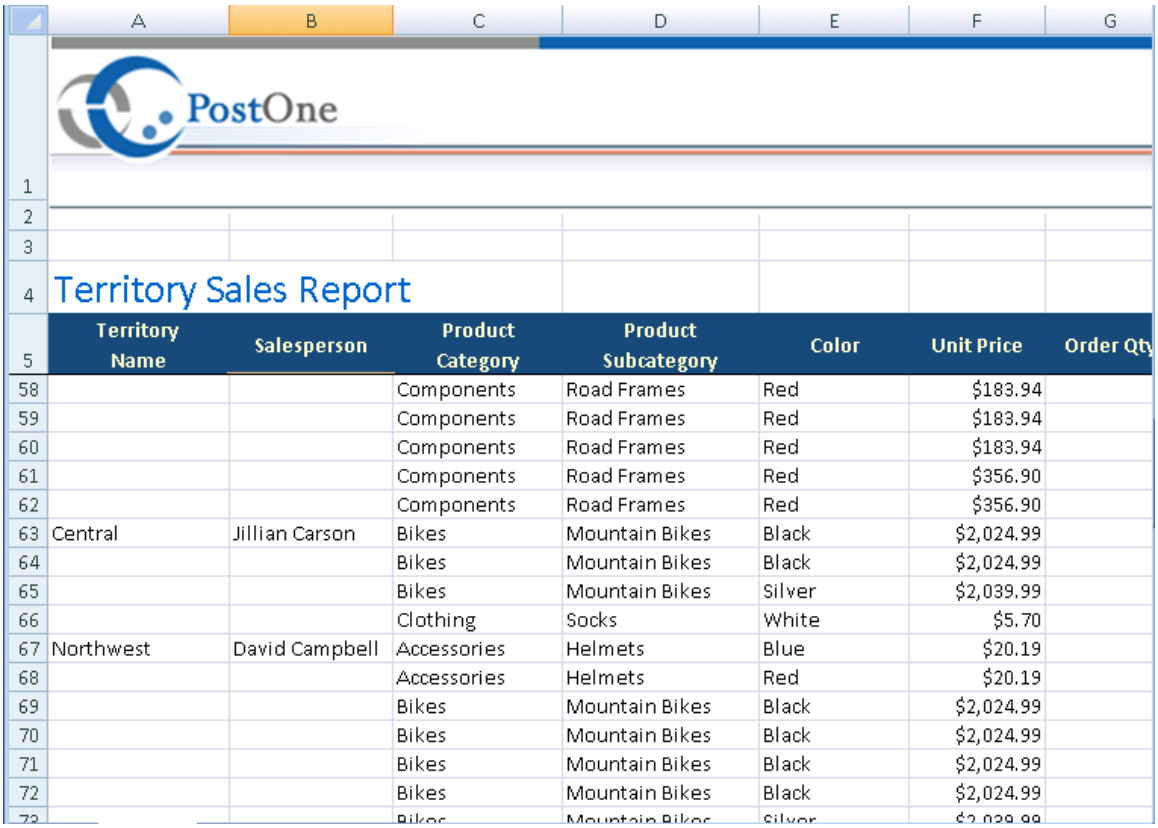
Nested grouping with multiple groups

You can group by multiple groups by placing extra **%%group** markers in more columns. Additional **%%endgroup** markers are unnecessary. Each of the new fields will be grouped in order of hierarchy, from left to right unless otherwise specified. For example, if you want to group first by Territory Name, then by Salesperson, place an additional **%%group** marker in the second column next to your first **%%group** marker:



	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
6	%%group	%%group					
7	%%=[AdventureWo	%%=[AdventureV	%%=[AdventureW	%%=[AdventureWor	%%=[Adventure	%%=[Adventu	%%=[Adve
8	%%endgroup						
9	**Grand Total**						

The resulting spreadsheet will be grouped first by Territory Name, then by Salesperson:



	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
58			Components	Road Frames	Red	$183.94	
59			Components	Road Frames	Red	$183.94	
60			Components	Road Frames	Red	$183.94	
61			Components	Road Frames	Red	$356.90	
62			Components	Road Frames	Red	$356.90	
63	Central	Jillian Carson	Bikes	Mountain Bikes	Black	$2,024.99	
64			Bikes	Mountain Bikes	Black	$2,024.99	
65			Bikes	Mountain Bikes	Silver	$2,039.99	
66			Clothing	Socks	White	$5.70	
67	Northwest	David Campbell	Accessories	Helmets	Blue	$20.19	
68			Accessories	Helmets	Red	$20.19	
69			Bikes	Mountain Bikes	Black	$2,024.99	
70			Bikes	Mountain Bikes	Black	$2,024.99	
71			Bikes	Mountain Bikes	Black	$2,024.99	
72			Bikes	Mountain Bikes	Black	$2,024.99	
73			Bikes	Mountain Bikes	Silver	$2,039.99	

You can group even further by adding additional **%%group** tags:

	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
6	%%group	%%group	%%group	%%group			
7	%%=[AdventureWo	%%=[AdventureV	%%=[AdventureW	%%=[AdventureWor	%%=[Adventure	%%=[Adventu	%%=[Adve
8	%%endgroup						
9	**Grand Total**						
10							

The resulting spreadsheet:

	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
6	Canada	José Saraiva	Accessories	Helmets	Black	$20.19	
7					Blue	$20.19	
8					Blue	$20.19	
9					Red	$20.19	
10			Bikes	Mountain Bikes	Black	$2,024.99	
11					Black	$2,024.99	
12					Black	$2,024.99	
13					Black	$2,024.99	
14					Silver	$2,039.99	
15				Road Bikes	Red	$2,146.96	
16					Red	$2,146.96	
17					Red	$2,146.96	
18					Red	$874.79	
19					Red	$874.79	
20					Red	$874.79	
21					Red	$874.79	

Additional Functionality

Additional functionality such as headers and footers, are explained in the Advanced Grouping and Nesting page.

Advanced Grouping and Nesting

Introduced in build 7.0.0.1261

Starting in version 7.1 of ExcelWriter, you can use grouping and nesting markers to display flat data in a grouped and nested format using the

ExcelTemplate object.

Introduction

The previous tutorial, Using Grouping and Nesting with ExcelTemplate, showed how to use the new **%%group** marker to group hierarchical data from a flat data set. In this tutorial, we expand upon the concept by introducing advanced features: headers and footers for each group, the **%%value()** marker to show data values in headers and footers, the **(hide)** modifier to hide data that is already shown in a header or footer, and formatting options.

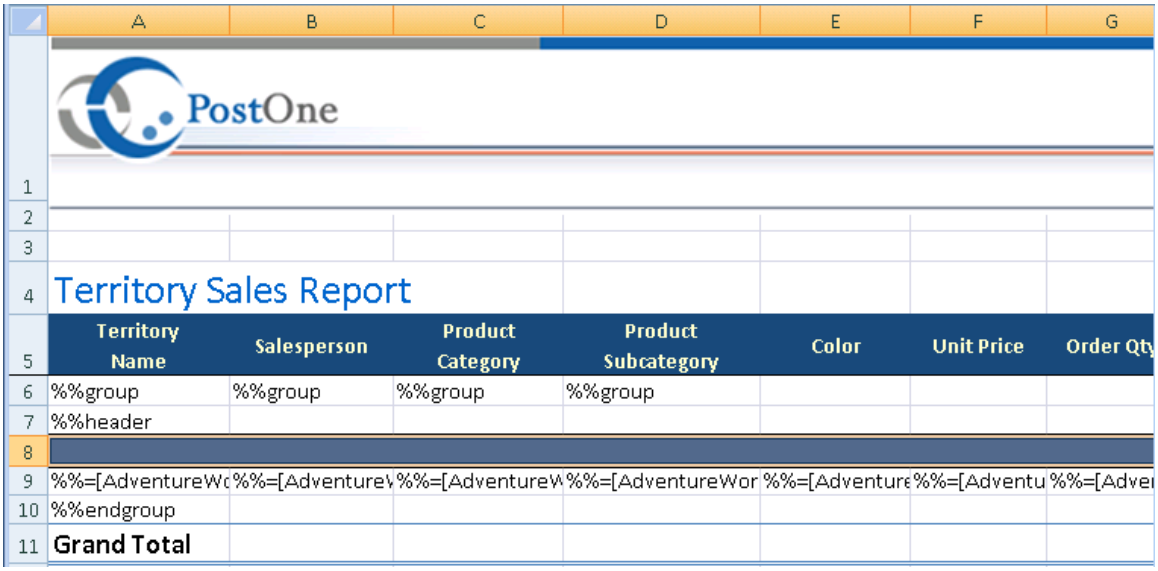
Setup

This tutorial uses the same code and data as the previous tutorial, Using Grouping and Nesting with ExcelTemplate. The template used in this tutorial starts where the previous tutorial left off.

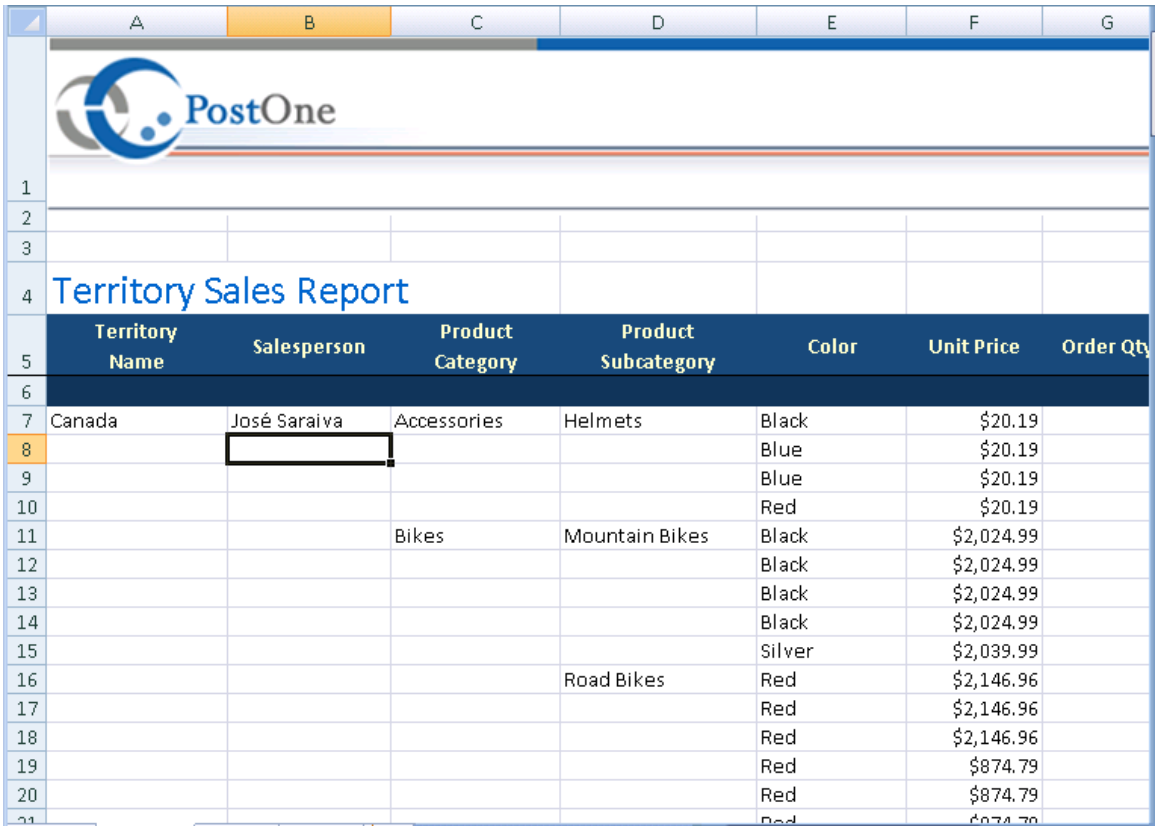
Adding Headers

Headers in ExcelWriter grouping and nesting are repeated for every new value in a grouped column. The beginning of a header block is indicated by a **%%header** marker, while the end of the block is marked by the next **%%header** block (if you have nested headers) or the data marker row.

The following template has a simple row of blue cells as the header for the Territory Name group:



When it is processed using the ExcelTemplate object, a row of blue cells is added above each new Territory Name block:



PostOne

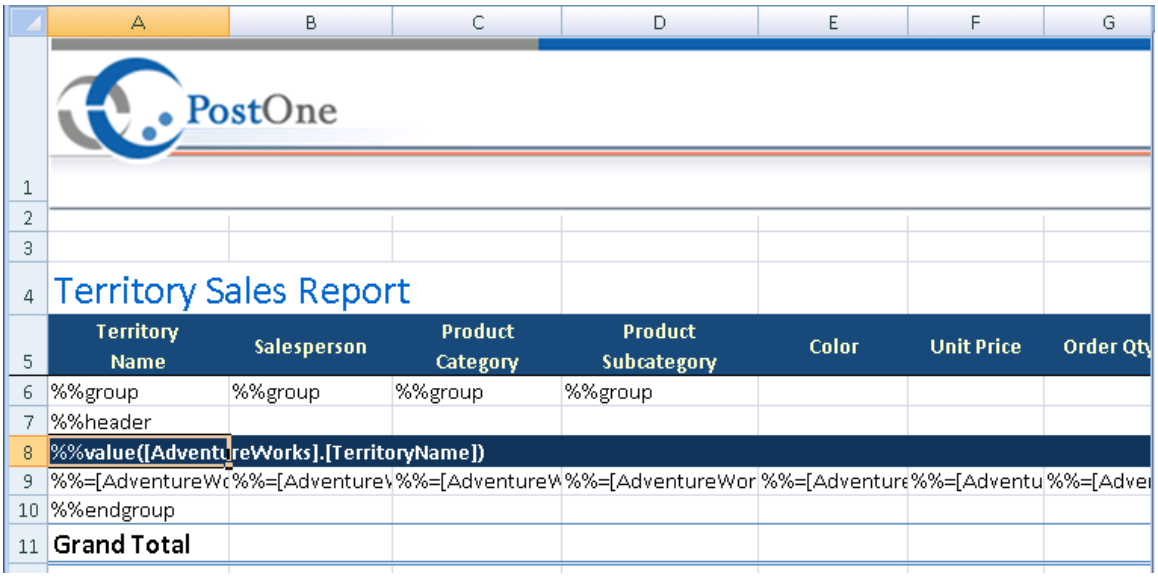
Territory Sales Report

Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
Canada	José Saraiva	Accessories	Helmets	Black	$20.19	
				Blue	$20.19	
				Blue	$20.19	
				Red	$20.19	
		Bikes	Mountain Bikes	Black	$2,024.99	
				Black	$2,024.99	
				Black	$2,024.99	
				Black	$2,024.99	
				Silver	$2,039.99	
			Road Bikes	Red	$2,146.96	
				Red	$2,146.96	
				Red	$2,146.96	
				Red	$874.79	
				Red	$874.79	

Using the %%value marker

While the above example helps to separate the different regions from each other in nested output, you can also add content to the headers. Any text that you place within header rows will be repeated as well for each header.

Static text within the header will be repeated unchanged for every header. You can dynamically add in data from the data rows using the **%%value** data marker. In the example below, we place each Territory Name in the header for each Territory Name by placing **%%value([AdventureWorks].[TerritoryName])** in the header. The text within the %%value() marker is the data marker you'd like to display, without the "%%=":

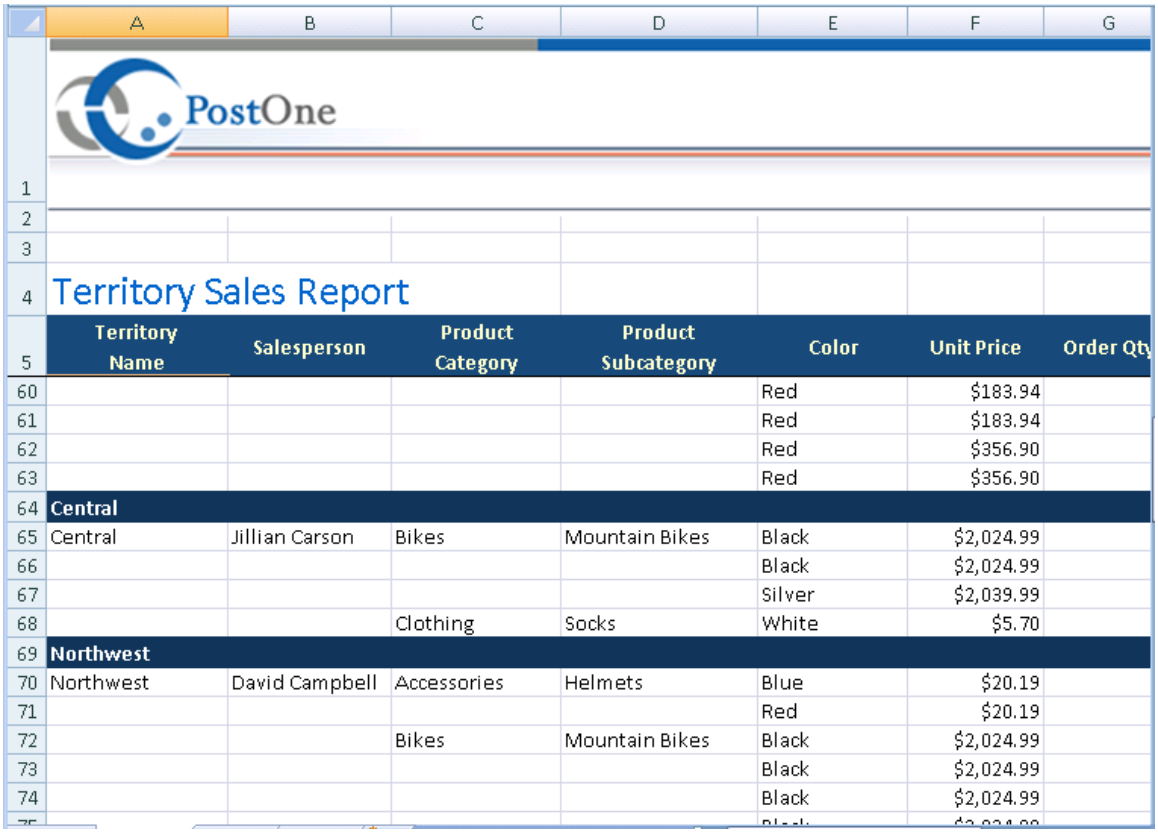


PostOne

Territory Sales Report

Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
%%group	%%group	%%group	%%group			
%%header						
%%value([AdventureWorks].[TerritoryName])						
%%=[AdventureWo	%%=[AdventureV	%%=[AdventureW	%%=[AdventureWor	%%=[Adventure	%%=[Adventu	%%=[Adve
%%endgroup						
Grand Total						

Each header will now feature the Territory Name of its group:



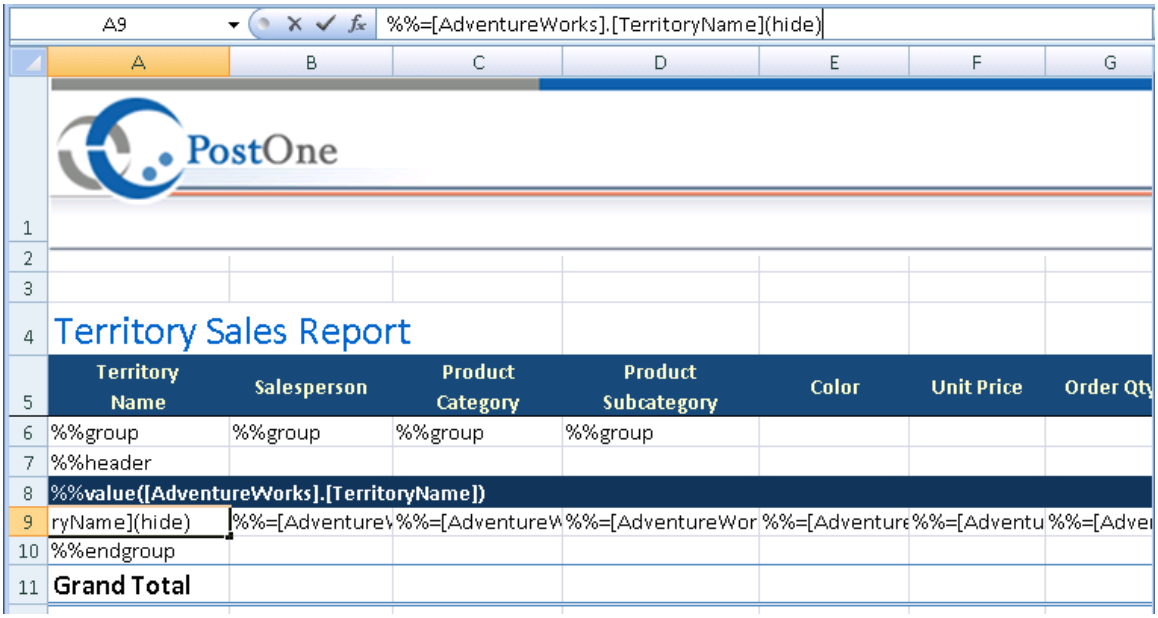
PostOne

Territory Sales Report

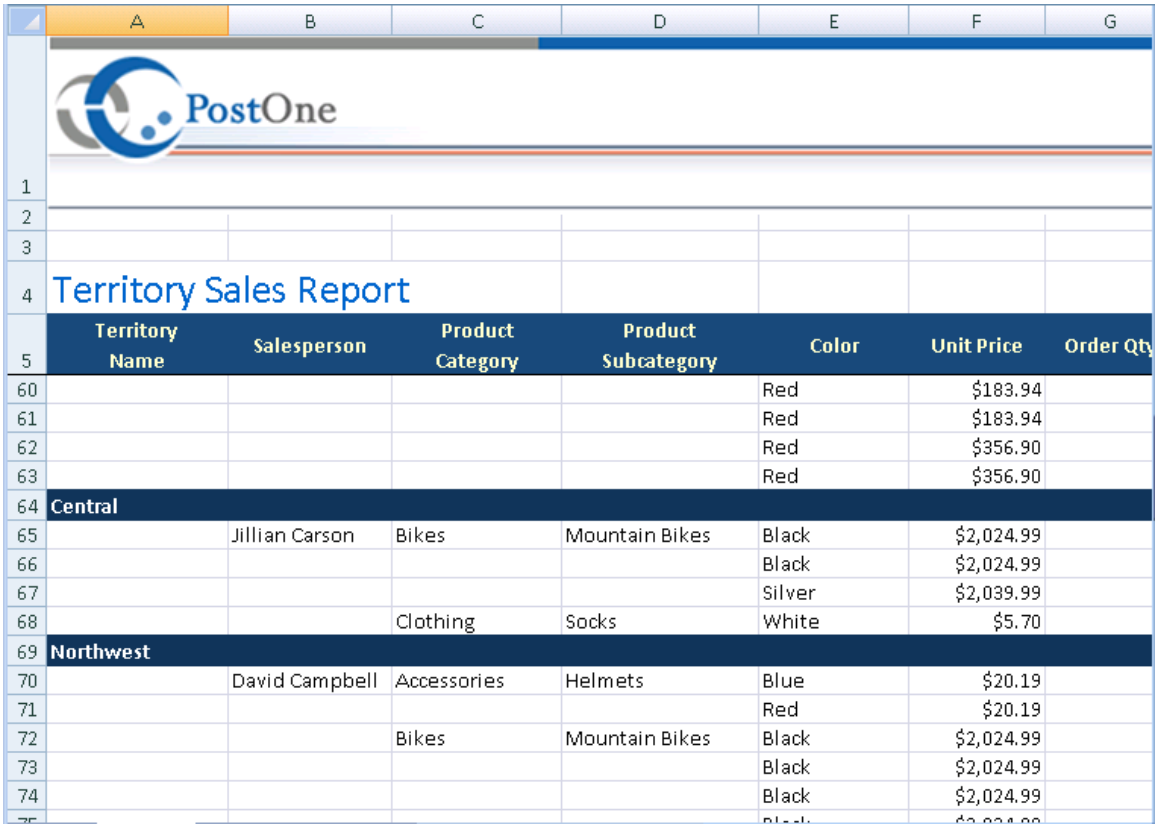
Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
				Red	$183.94	
				Red	$183.94	
				Red	$356.90	
				Red	$356.90	
Central						
Central	Jillian Carson	Bikes	Mountain Bikes	Black	$2,024.99	
				Black	$2,024.99	
				Silver	$2,039.99	
		Clothing	Socks	White	$5.70	
Northwest						
Northwest	David Campbell	Accessories	Helmets	Blue	$20.19	
				Red	$20.19	
		Bikes	Mountain Bikes	Black	$2,024.99	
				Black	$2,024.99	
				Black	$2,024.99	

Using the (hide) data marker modifier

In the example above, we were able to place the Territory Name values in the header using the **%%value** marker. However, the Territory Name values are repeated still in the data rows. You can hide the repeated values in the data marker row by appending the **(hide)** data marker modifier to the end of the **%%=[AdventureWorks].[TerritoryName]** data marker in the data marker row:



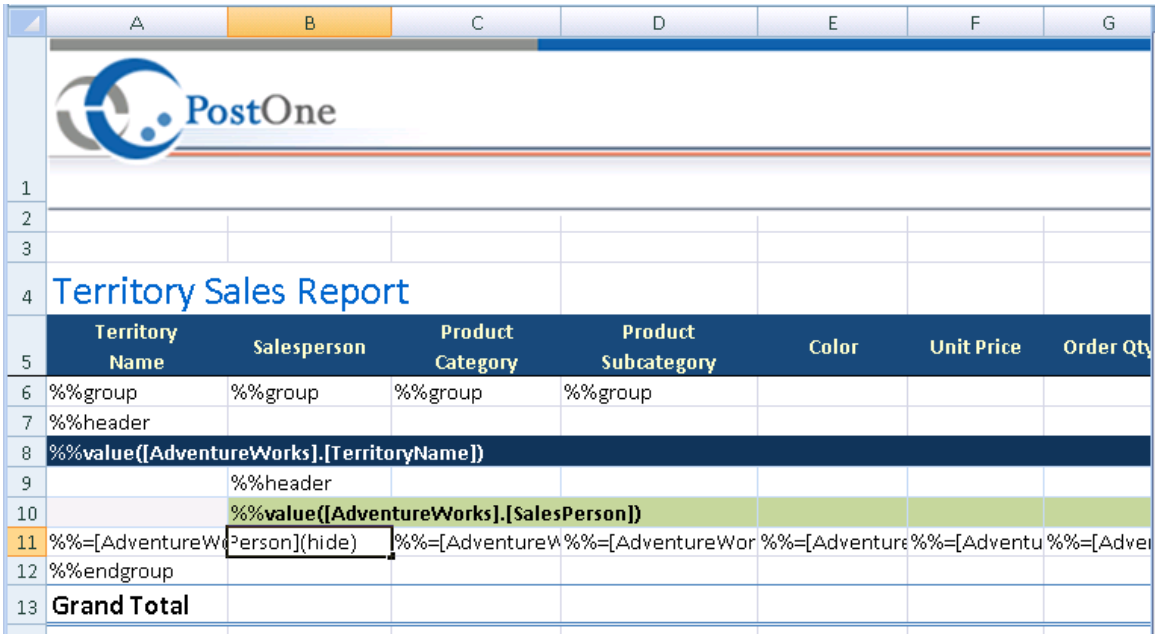
When the data is bound to the template, the Territory Name field is omitted from the data rows, while it is preserved in the headers:



	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
60					Red	$183.94	
61					Red	$183.94	
62					Red	$356.90	
63					Red	$356.90	
64	Central						
65		Jillian Carson	Bikes	Mountain Bikes	Black	$2,024.99	
66					Black	$2,024.99	
67					Silver	$2,039.99	
68			Clothing	Socks	White	$5.70	
69	Northwest						
70		David Campbell	Accessories	Helmets	Blue	$20.19	
71					Red	$20.19	
72			Bikes	Mountain Bikes	Black	$2,024.99	
73					Black	$2,024.99	
74					Black	$2,024.99	

Nesting Headers

In the examples above, we define a header for the top field in the hierarchy directly above the data marker rows. You can further organize your data using headers for nested fields. To do so, add a **%%header** marker to the field's column in the row directly below the header that precedes it. In the example below, we add a header to the Salesperson field. The header has a green background and includes the value of the field. We've also removed that value from the data rows using the **(hide)** data marker modifier:



	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
6	%%group	%%group	%%group	%%group			
7	%%header						
8	%%value([AdventureWorks].[TerritoryName])						
9		%%header					
10		%%value([AdventureWorks].[SalesPerson])					
11	%%=[AdventureW	Person](hide)	%%=[AdventureW	%%=[AdventureWor	%%=[Adventure	%%=[Adventu	%%=[Adve
12	%%endgroup						
13	Grand Total						

When the template is filled, the Salesperson headers are repeated for every Salesperson within their corresponding Territory groups:

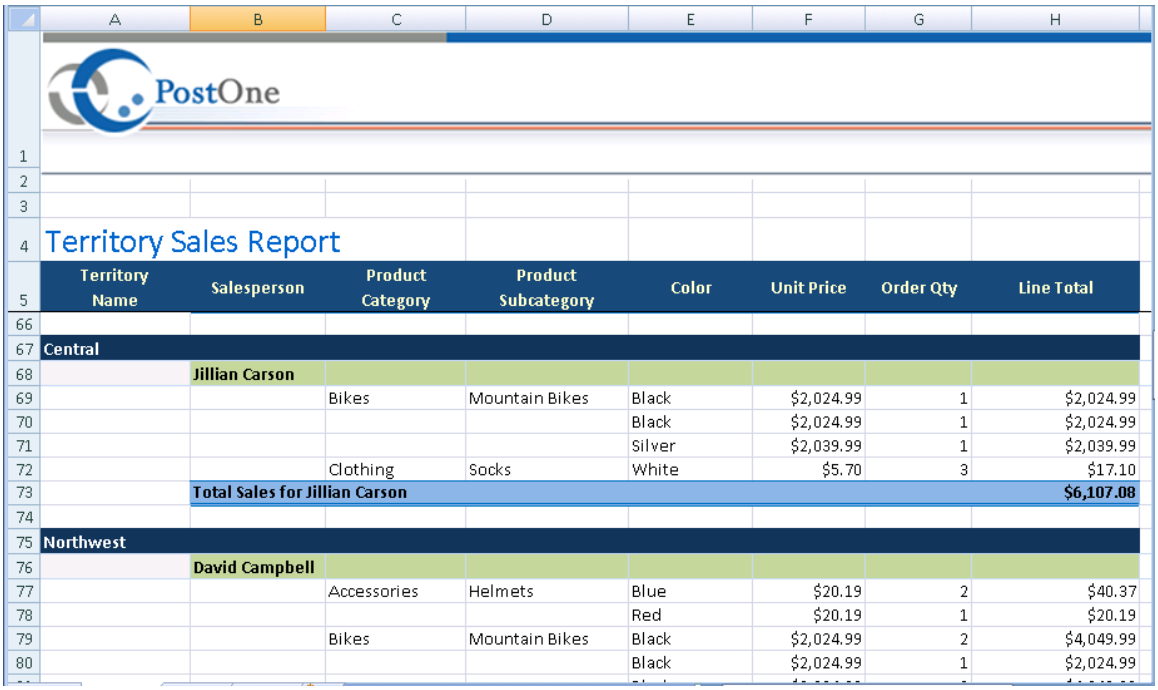
	A	B	C	D	E	F	G
1	PostOne						
2							
3							
4	Territory Sales Report						
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty
63					Red	$356.90	
64					Red	$356.90	
65	Central						
66		Jillian Carson					
67			Bikes	Mountain Bikes	Black	$2,024.99	
68					Black	$2,024.99	
69					Silver	$2,039.99	
70			Clothing	Socks	White	$5.70	
71	Northwest						
72		David Campbell					
73			Accessories	Helmets	Blue	$20.19	
74					Red	$20.19	
75			Bikes	Mountain Bikes	Black	$2,024.99	
76					Black	$2,024.99	
77					Black	$2,024.99	

Adding Footers

The **%%footer** marker allows you to add a footer for your groups. The first footer should be placed directly below the data row and will end with either the next **%%footer** marker or the **%%endgroup** marker. Everything in the footer will be repeated below the data for each value in the given field. In the example below, we've added a footer to the Salesperson field. It contains two rows: a blank row and a row in color blue using the formula **="Total Sales For " & B10** in cell B13 and the formula **=SUBTOTAL(9, H11:H11)** in cell H13.

	A	B	C	D	E	F	G	H
1	PostOne							
2								
3								
4	Territory Sales Report							
5	Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty	Line Total
6	%%group	%%group	%%group	%%group				
7	%%header							
8	%%value([AdventureWorks].[TerritoryName])							
9		%%header						
10		%%value([AdventureWorks].[SalesPerson])						
11	%%=[AdventureWo	%%=[AdventureV	%%=[AdventureW	%%=[AdventureWor	%%=[Adventure	%%=[Adventu	%%=[Adventu	%%=[AdventureWorks]
12		%%footer						
13		Total Sales For %%	value([AdventureWorks].[SalesPerson])					$0.00
14								
15	%%endgroup							
16	Grand Total							$0.00

The formula in cell B13 will evaluate to say "Total Sales For [Salesperson]" for each salesperson. The formula in cell H13 will subtotal the sums of each Line Total for each salesperson:



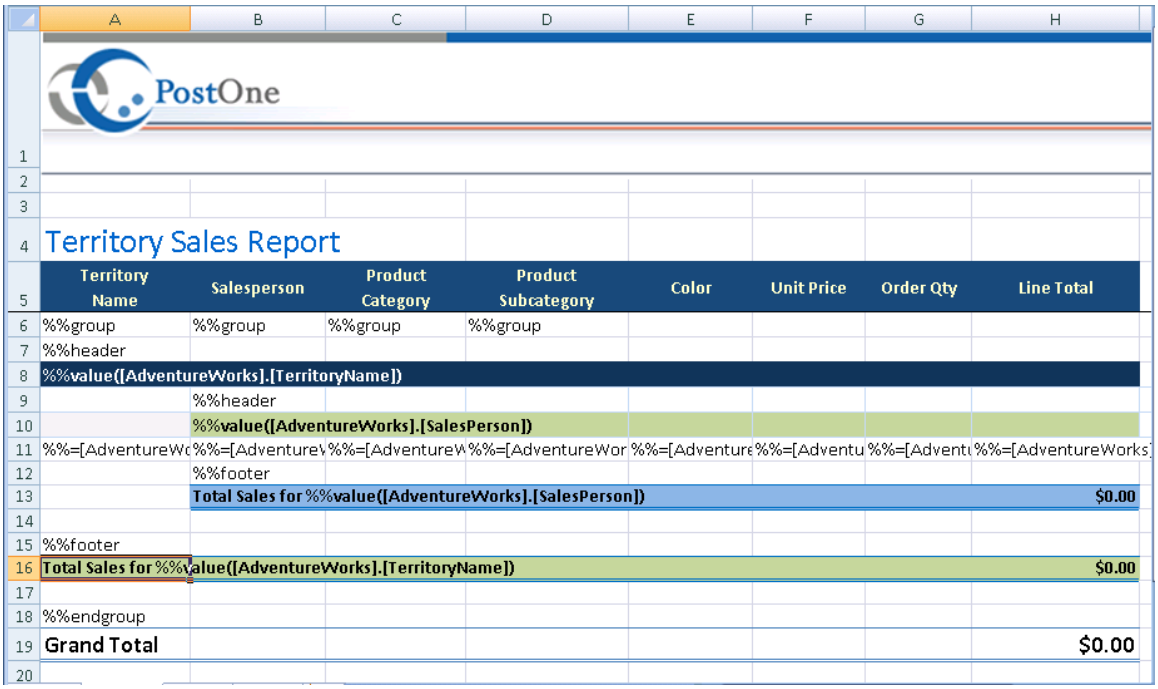
PostOne

Territory Sales Report

Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty	Line Total
Central							
	Jillian Carson						
		Bikes	Mountain Bikes	Black	$2,024.99	1	$2,024.99
				Black	$2,024.99	1	$2,024.99
				Silver	$2,039.99	1	$2,039.99
		Clothing	Socks	White	$5.70	3	$17.10
	Total Sales for Jillian Carson						$6,107.08
Northwest							
	David Campbell						
		Accessories	Helmets	Blue	$20.19	2	$40.37
				Red	$20.19	1	$20.19
		Bikes	Mountain Bikes	Black	$2,024.99	2	$4,049.99
				Black	$2,024.99	1	$2,024.99

Nesting Footers

Just as with headers, footers can be nested within each other. If you place a **%%footer** marker for another column, the results will be nested within each other just as they were with the headers. In this case, we will add a footer for the Territory Name column directly below the Salesperson footer. This time, cell A16 will have the formula **="Total Sales for " & A8** and cell H16 will have formula **=SUBTOTAL(9, H11:H11)** :

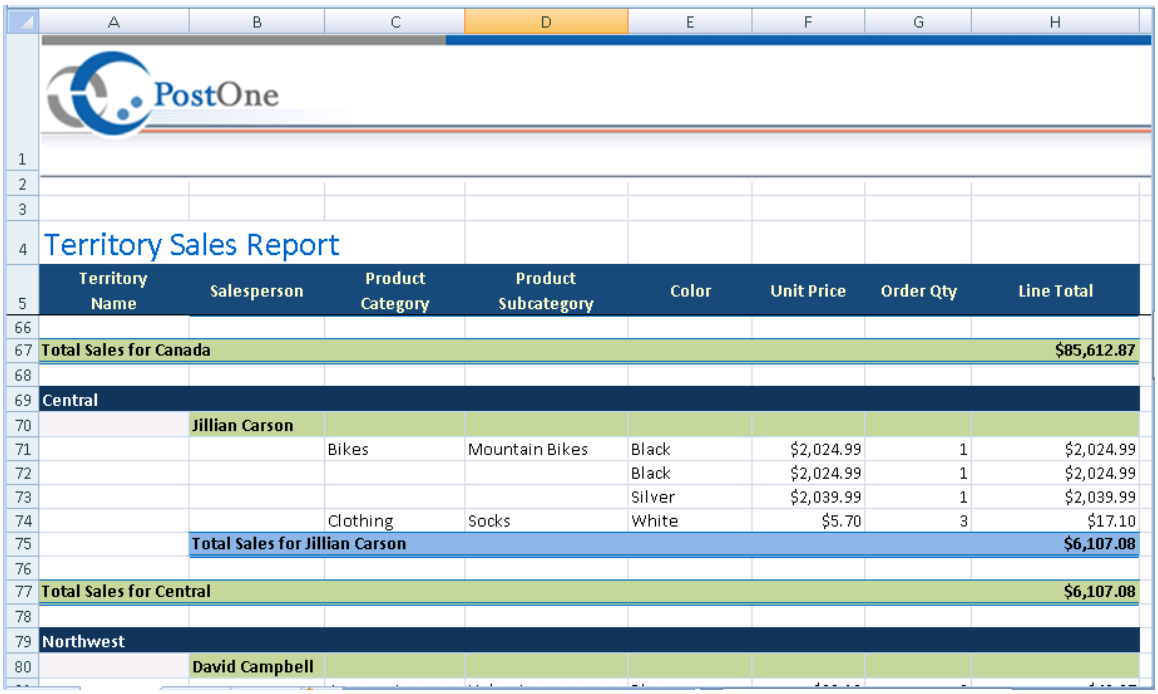


PostOne

Territory Sales Report

Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty	Line Total
%%group	%%group	%%group	%%group				
%%header							
%%value([AdventureWorks].[TerritoryName])							
	%%header						
	%%value([AdventureWorks].[SalesPerson])						
%%=[AdventureWo	%%=[AdventureV	%%=[AdventureW	%%=[AdventureWor	%%=[Adventure	%%=[Adventu	%%=[Adventi	%%=[AdventureWorks]
	%%footer						
	Total Sales for %%value([AdventureWorks].[SalesPerson])						$0.00
%%footer							
Total Sales for %%value([AdventureWorks].[TerritoryName])							$0.00
%%endgroup							
Grand Total							$0.00

Column A of the footer will evaluate to say "Total Sales for [Territory Name]", while column H of the footer will subtotal the values of each Line Total for each Territory Name. Because we used the **SUBTOTAL** formula instead of the **SUM** formula, the subtotals of our Salesperson footer will not be included in the sum:



PostOne

Territory Sales Report

Territory Name	Salesperson	Product Category	Product Subcategory	Color	Unit Price	Order Qty	Line Total
Total Sales for Canada							$85,612.87
Central							
	Jillian Carson						
		Bikes	Mountain Bikes	Black	$2,024.99	1	$2,024.99
				Black	$2,024.99	1	$2,024.99
				Silver	$2,039.99	1	$2,039.99
		Clothing	Socks	White	$5.70	3	$17.10
	Total Sales for Jillian Carson						$6,107.08
Total Sales for Central							$6,107.08
Northwest							
	David Campbell						

Creating a Grouped and Nested Spreadsheet

Introduced in build 7.0.0.1261

Starting in version 7.1 of ExcelWriter, you can use grouping and nesting markers to display flat data in a grouped and nested format using the ExcelTemplate object.

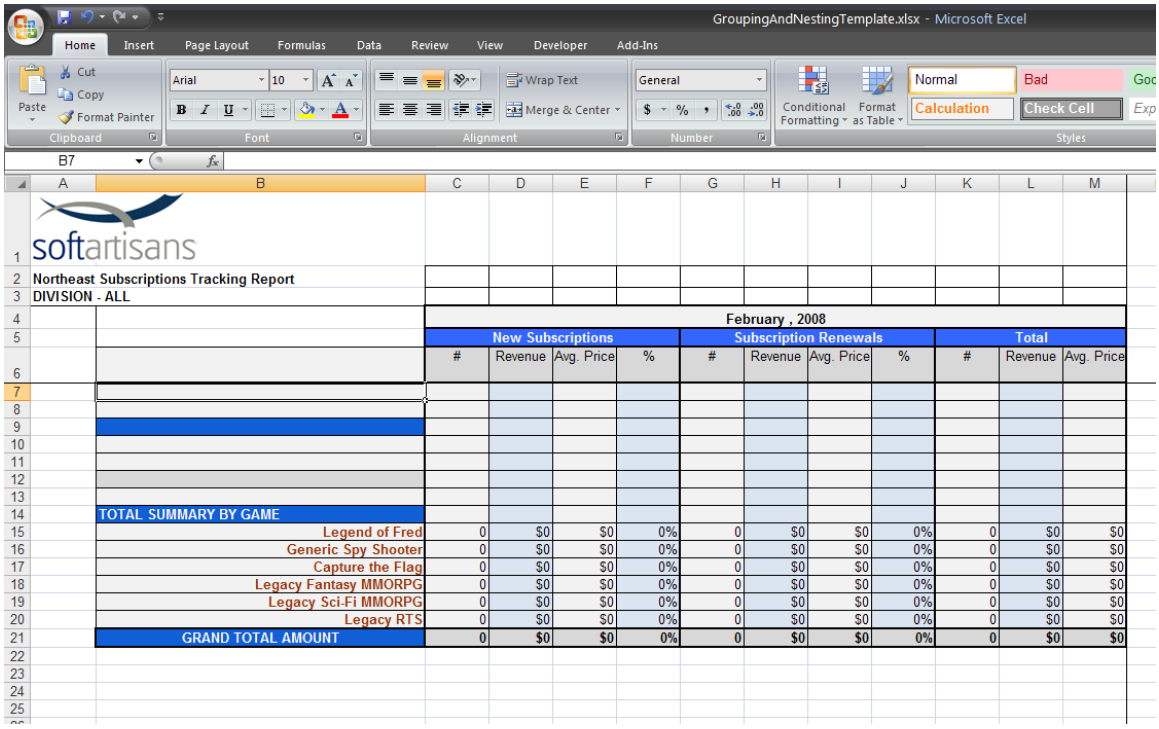
Code Sample: Creating a Grouping and Nested Spreadsheet

[C#]

Screenshots from the sample are used below.

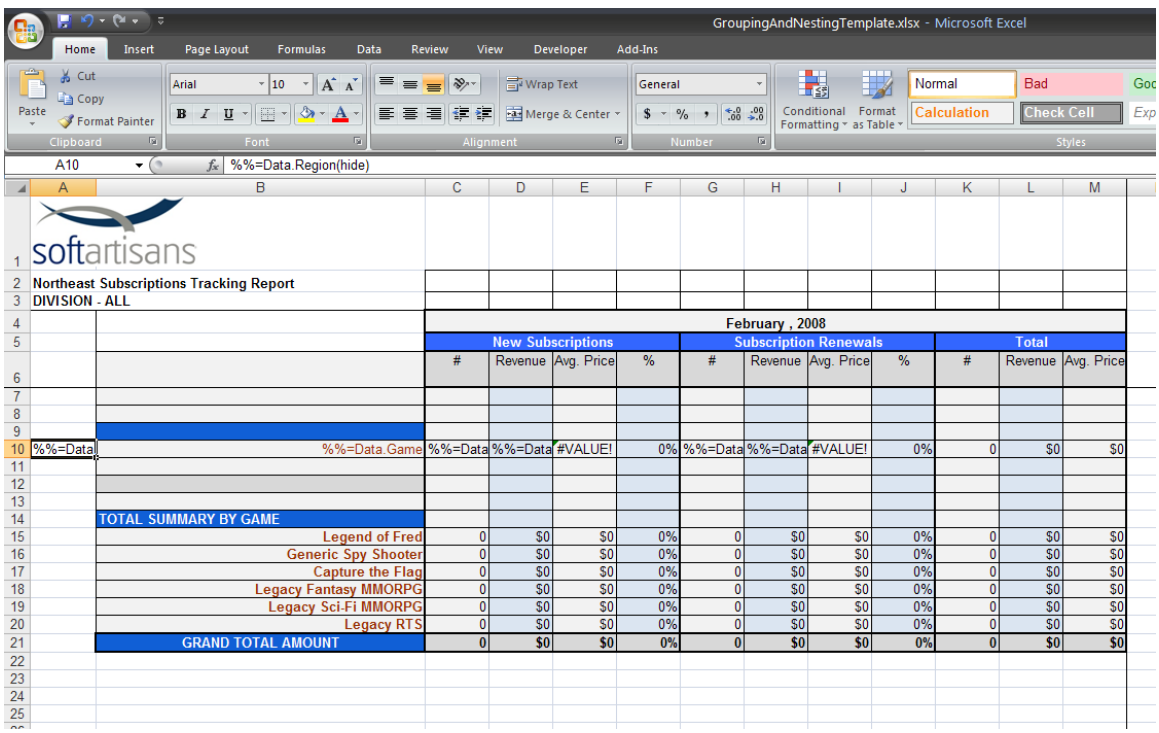
How to create a template spreadsheet with grouping functionality

1. In Microsoft Excel, open an Office Open XML (.xlsx) spreadsheet to use as a template.



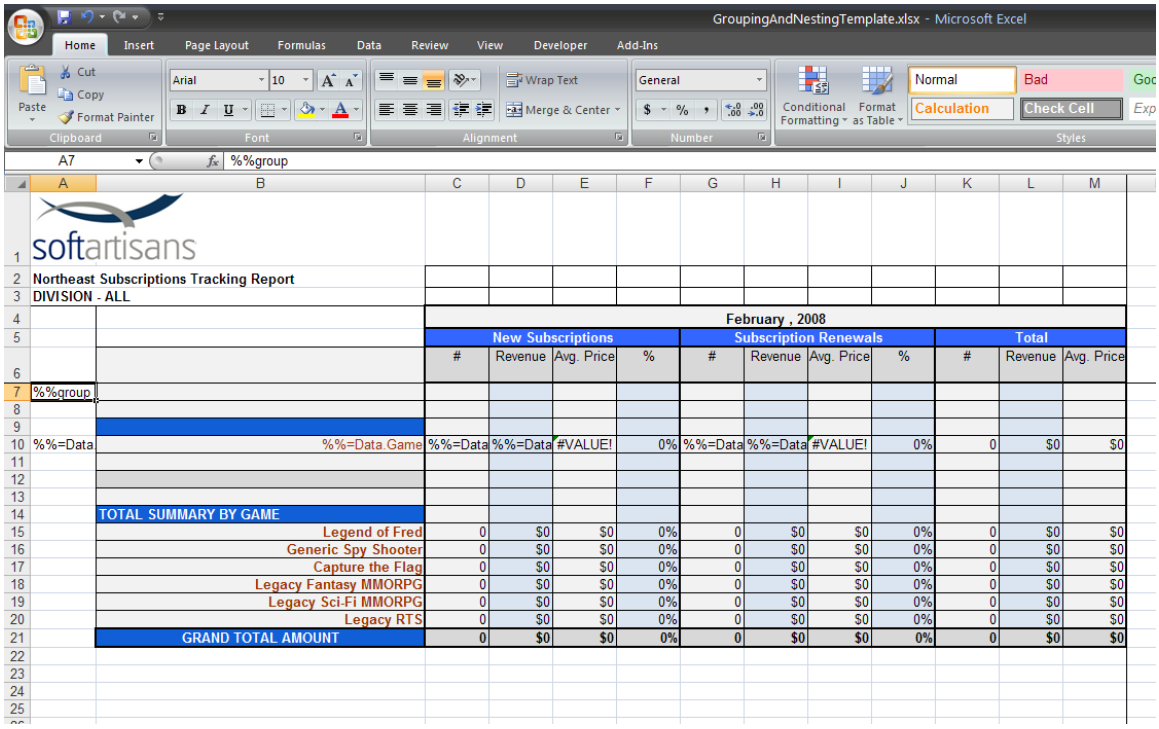
1	softartisans										
2	Northeast Subscriptions Tracking Report										
3	DIVISION - ALL										
4		February , 2008									
5		New Subscriptions				Subscription Renewals				Total		
6		#	Revenue	Avg. Price	%	#	Revenue	Avg. Price	%	#	Revenue	Avg. Price
14	TOTAL SUMMARY BY GAME											
15	Legend of Fred	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
16	Generic Spy Shooter	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
17	Capture the Flag	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
18	Legacy Fantasy MMORPG	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
19	Legacy Sci-Fi MMORPG	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
20	Legacy RTS	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
21	GRAND TOTAL AMOUNT	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0

2. In the template worksheet, create a data marker row containing the fields you wish to bind to the template. In this case, a **(hide)** data marker modifier is added to the **%%=Data.Region** data marker so the grouping values are not repeated for every row of data. Additionally, several of the cells of this row have formulas that references cells within the row.

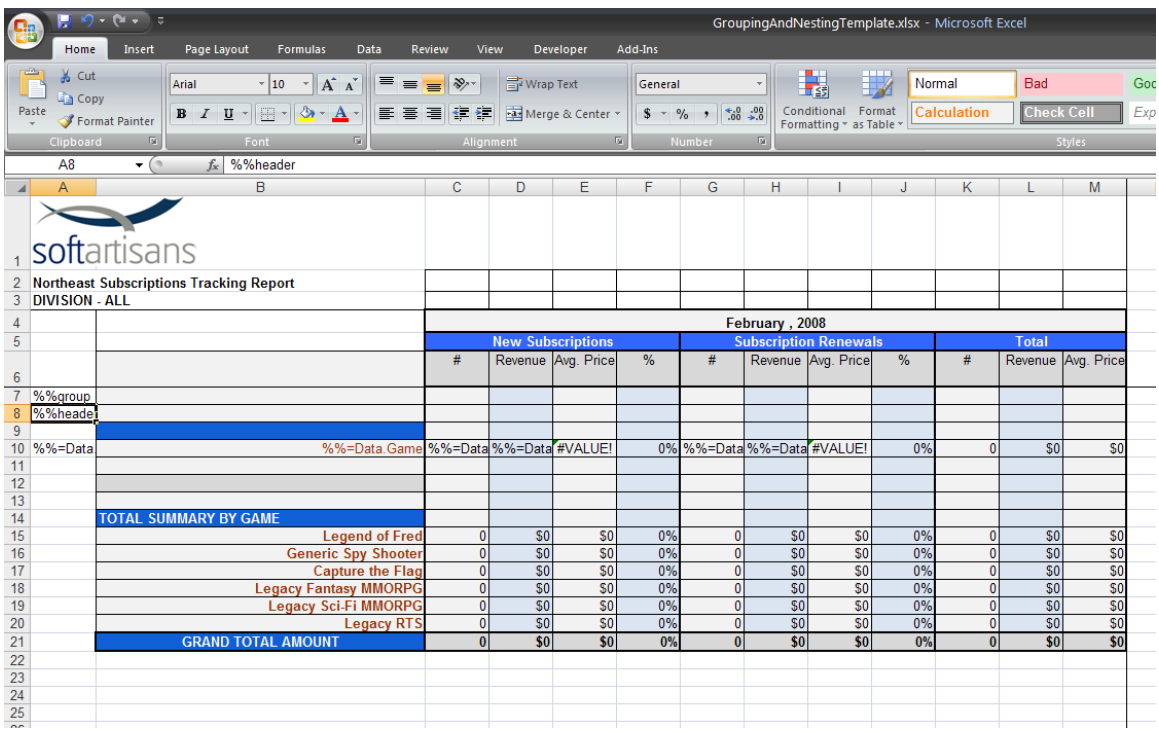


1	softartisans											
2	Northeast Subscriptions Tracking Report											
3	DIVISION - ALL											
4		February , 2008										
5		New Subscriptions				Subscription Renewals				Total		
6		#	Revenue	Avg. Price	%	#	Revenue	Avg. Price	%	#	Revenue	Avg. Price
10	%%=Data	%%=Data.Game	%%=Data	%%=Data	#VALUE!	0%	%%=Data	%%=Data	#VALUE!	0%	0	$0	$0
14	TOTAL SUMMARY BY GAME											
15	Legend of Fred	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
16	Generic Spy Shooter	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
17	Capture the Flag	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
18	Legacy Fantasy MMORPG	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
19	Legacy Sci-Fi MMORPG	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
20	Legacy RTS	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0
21	GRAND TOTAL AMOUNT	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0

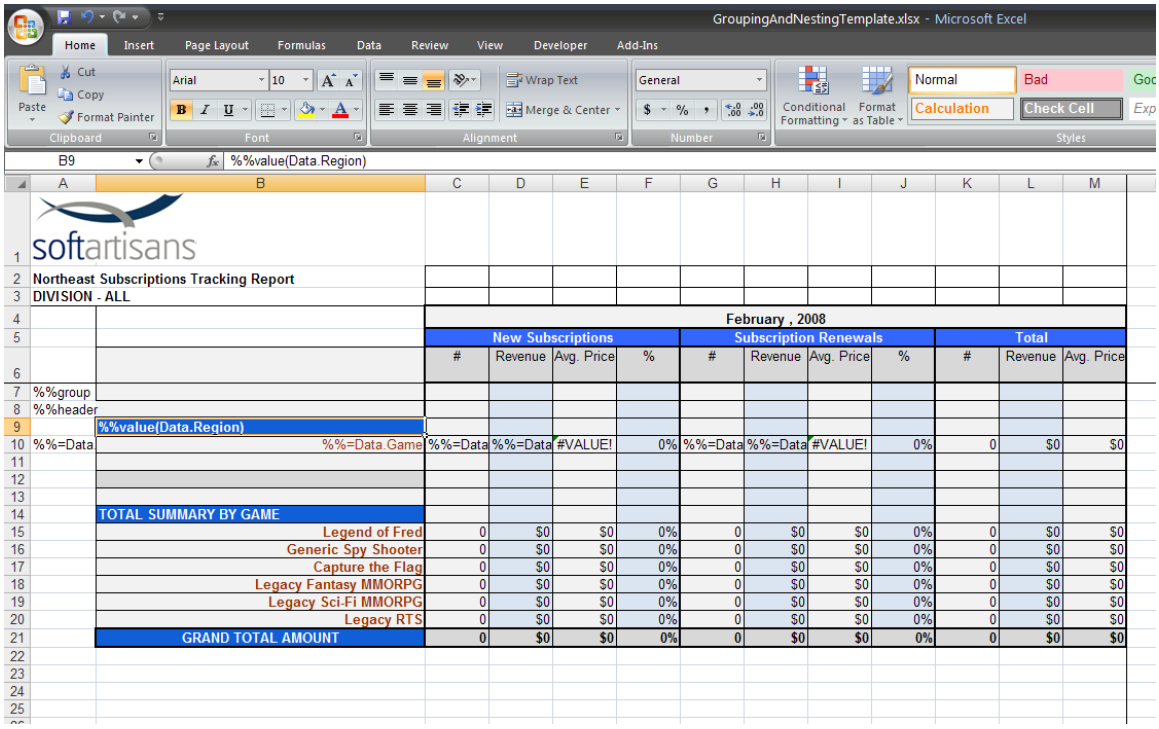
3. In a row above the field by which you wish to group, insert a **%%group** marker to indicate that you are grouping by that field, in this case the Data.Region field.



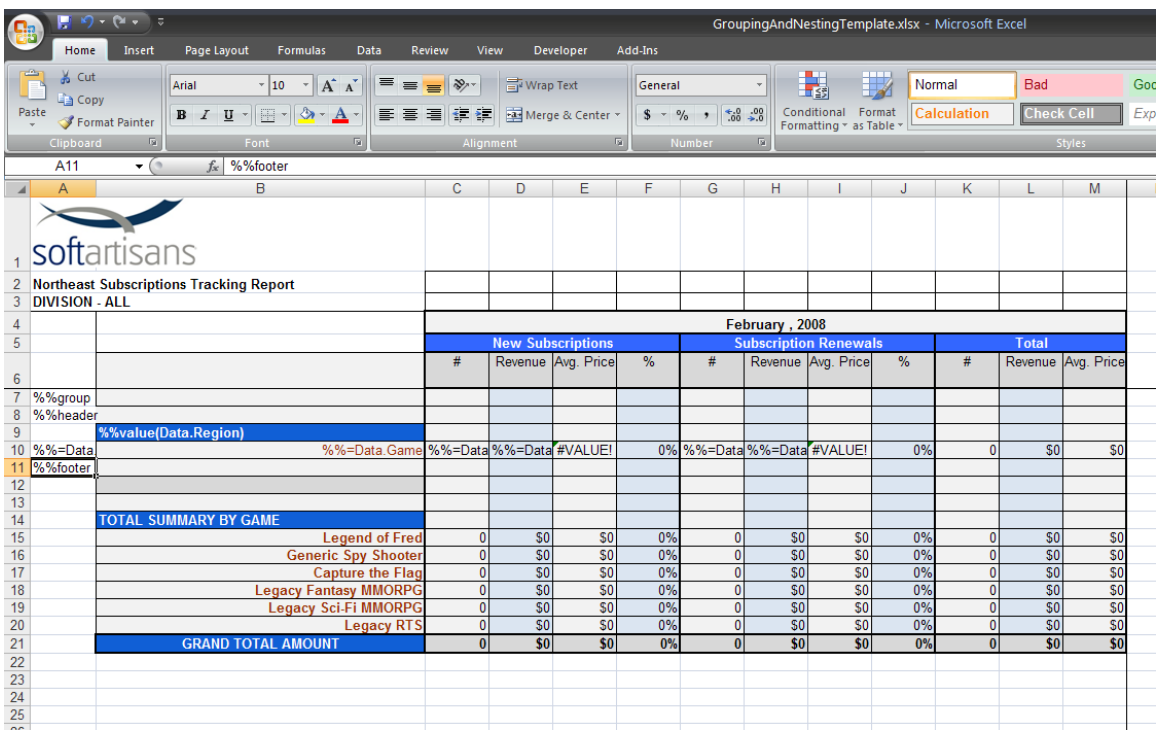
4. In the cell directly below the one containing the new **%%group** marker, place a **%%header** marker to indicate the beginning of the header rows.



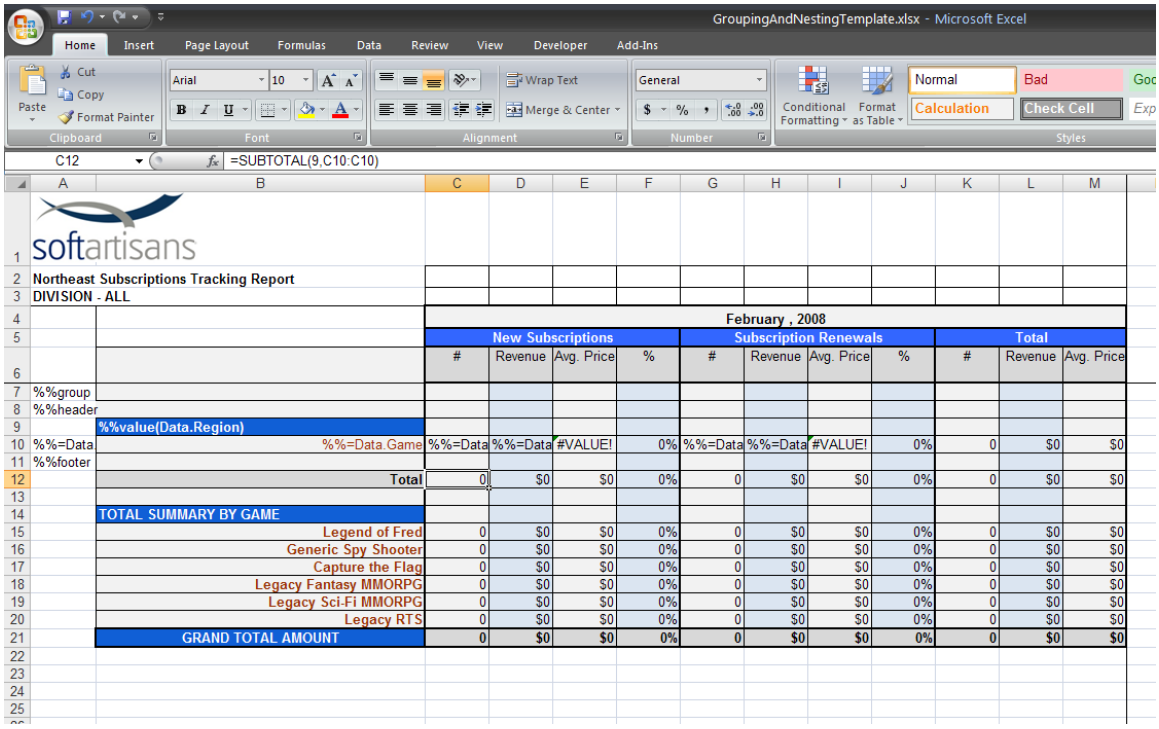
5. In the rows between the **%%header** marker and your data marker row, put content that you wish to see repeated each time there is a new value in the grouping field. In this spreadsheet, the value of the Data.Region is in the header above the rows associated with that region.



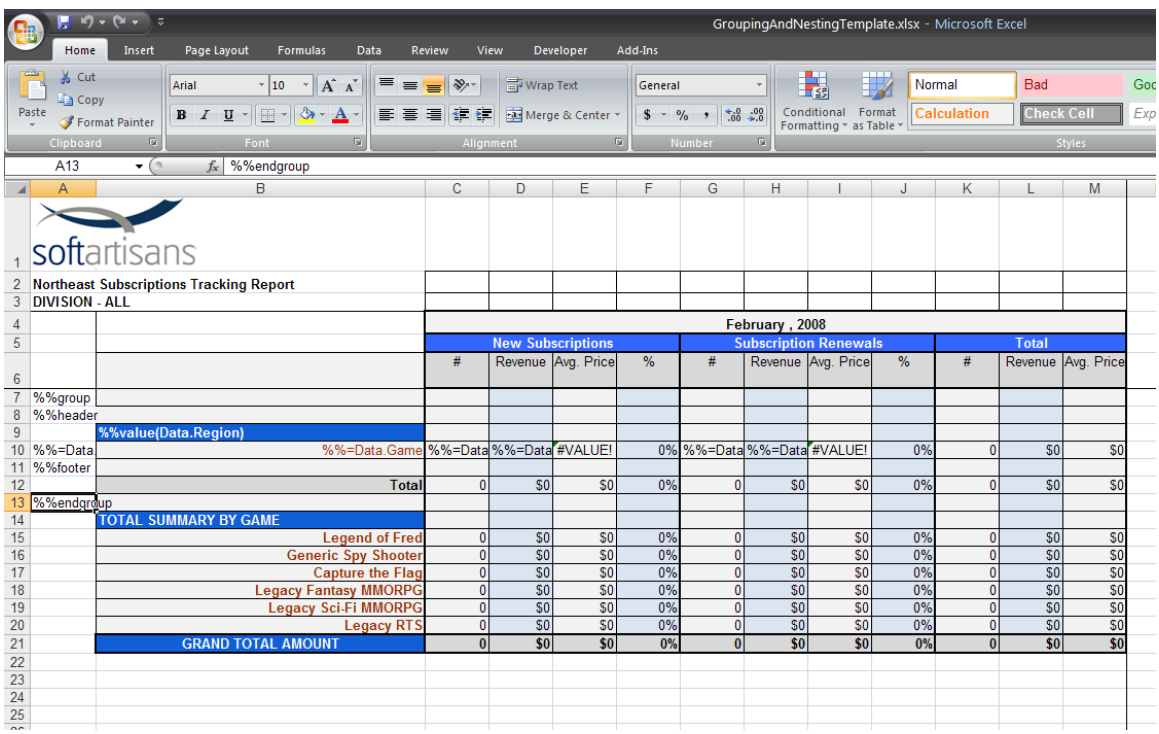
6. In the cell directly underneath the grouping field, place a **%%footer** marker to indicate the beginning of the footer rows.



7. In the rows under the **%%footer** marker, put content that you wish to see repeated each time there is a new value in the grouping field. In this case, a subtotal row calculates subtotals for each group.



8. Under the footer rows, place an **%%endgroup** marker to indicate the end of the grouping block.



Input Data

Generally, there are few changes that need to be made to data to use ExcelTemplate's Grouping and Nesting functionality. There are some guidelines, however, that you must follow when setting up your data source:

- Data must be presorted in a hierarchic manner. Because the grouping and nesting code essentially removes repeated values from the grouping column, the grouping column values can be repeated if the data are not already sorted.
- Data must be in a flat format – that is, there should be a value in every row and column of the input data set.

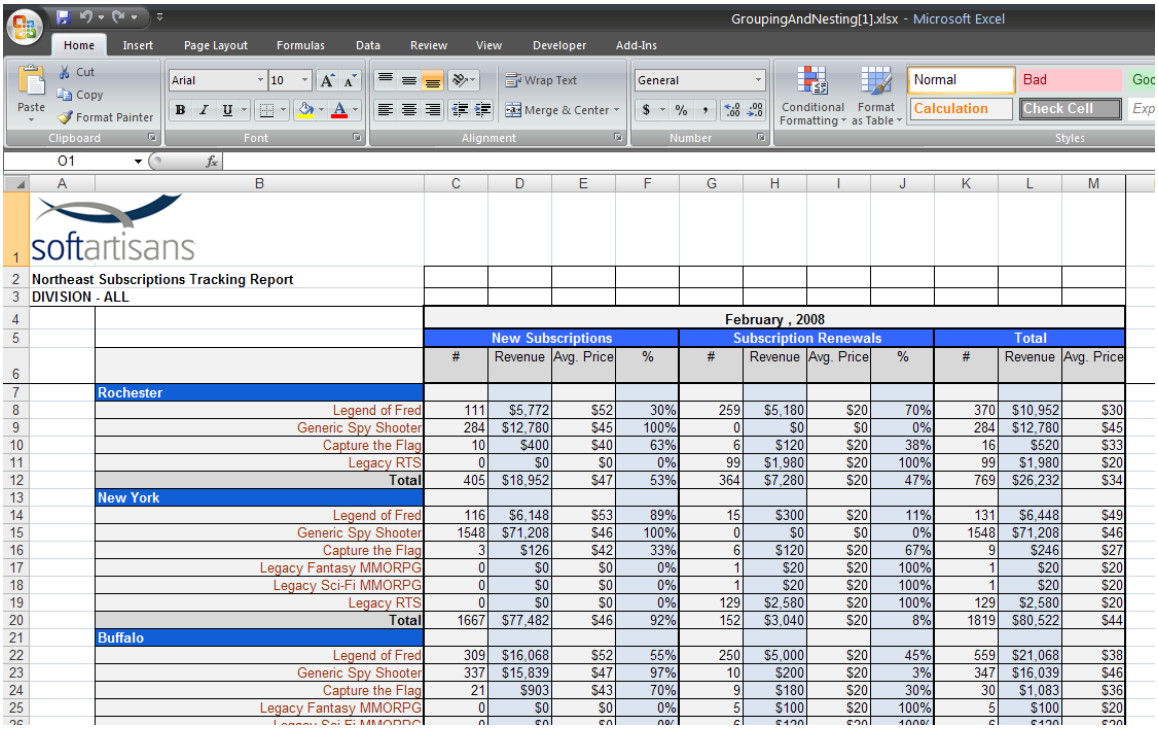
Data Binding Code

Bind data to a grouping block just as you would group it to a normal data marker row. Use the ExcelTemplate.BindData() method to bind the data to the spreadsheet. The following code opens a spreadsheet with a grouping block, creates a DataSet from an XML file, binds the DataSet to the template, then processes the file and streams it to the user.

```
private void GenerateReport()
{
     //--- Create a new ExcelTemplate object and open a template file
     //--- (Office Open XML format -- .xlsx) containing grouping data markers
     ExcelTemplate xlt = new ExcelTemplate();
     xlt.Open(Server.MapPath("templates/GroupingAndNestingTemplate.xlsx"));

     //--- Create a flat data source, sorted based on the grouping categories
     DataSet ds = new DataSet();
     ds.ReadXml(Server.MapPath("VideoGameData.xml"), XmlReadMode.ReadSchema);

     //--- Bind the data to the template file and send the resulting file
     //--- to the client.  Office Open XML (.xlsx) format must be used.
     xlt.BindData(ds, "Data", xlt.CreateDataBindingProperties());
     xlt.Process();
     xlt.Save(Page.Response, "GroupingAndNesting.xlsx", false);
}
```

Output file

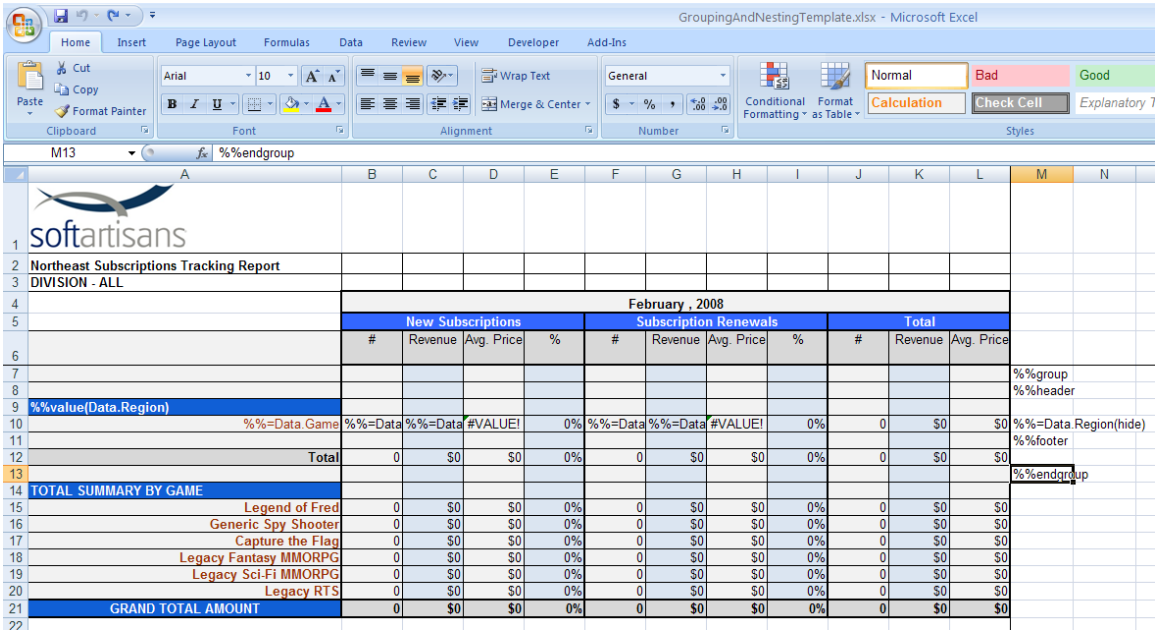
Below are shown the first few groups from the output file. Notice that the group, header, footer, and endgroup marker rows are not shown in the output file, while the hidden Data.Region column data is also removed.



softartisans											
Northeast Subscriptions Tracking Report											
DIVISION - ALL											
	February , 2008										
	New Subscriptions				Subscription Renewals				Total		
	#	Revenue	Avg. Price	%	#	Revenue	Avg. Price	%	#	Revenue	Avg. Price
Rochester											
Legend of Fred	111	$5,772	$52	30%	259	$5,180	$20	70%	370	$10,952	$30
Generic Spy Shooter	284	$12,780	$45	100%	0	$0	$0	0%	284	$12,780	$45
Capture the Flag	10	$400	$40	63%	6	$120	$20	38%	16	$520	$33
Legacy RTS	0	$0	$0	0%	99	$1,980	$20	100%	99	$1,980	$20
Total	405	$18,952	$47	53%	364	$7,280	$20	47%	769	$26,232	$34
New York											
Legend of Fred	116	$6,148	$53	89%	15	$300	$20	11%	131	$6,448	$49
Generic Spy Shooter	1548	$71,208	$46	100%	0	$0	$0	0%	1548	$71,208	$46
Capture the Flag	3	$126	$42	33%	6	$120	$20	67%	9	$246	$27
Legacy Fantasy MMORPG	0	$0	$0	0%	1	$20	$20	100%	1	$20	$20
Legacy Sci-Fi MMORPG	0	$0	$0	0%	1	$20	$20	100%	1	$20	$20
Legacy RTS	0	$0	$0	0%	129	$2,580	$20	100%	129	$2,580	$20
Total	1667	$77,482	$46	92%	152	$3,040	$20	8%	1819	$80,522	$44
Buffalo											
Legend of Fred	309	$16,068	$52	55%	250	$5,000	$20	45%	559	$21,068	$38
Generic Spy Shooter	337	$15,839	$47	97%	10	$200	$20	3%	347	$16,039	$46
Capture the Flag	21	$903	$43	70%	9	$180	$20	30%	30	$1,083	$36
Legacy Fantasy MMORPG	0	$0	$0	0%	5	$100	$20	100%	5	$100	$20

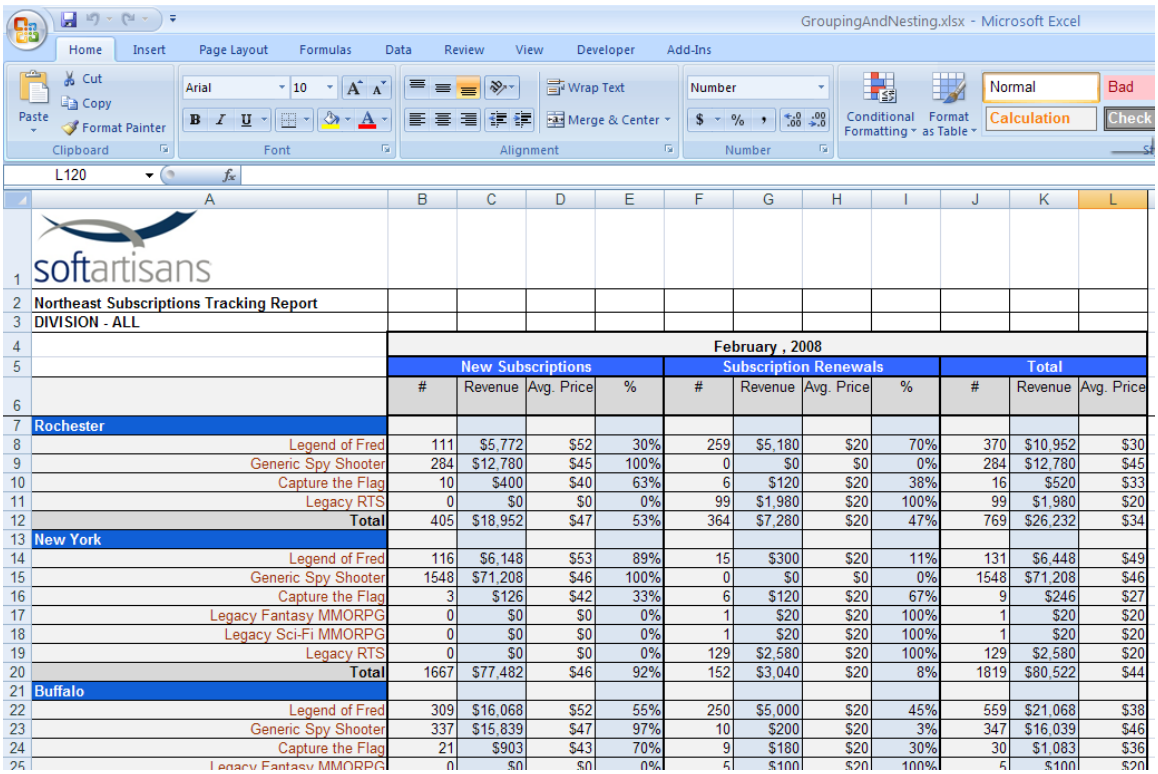
Formatting Options

ExcelWriter's Grouping and Nesting formatting is flexible enough to allow for a variety of different layouts. The **%%group**, **%%header**, and **%%footer** markers must be in the same column as the data column you wish to group, but there is no restriction for where that column must be. If you'd like to put the group data directly below the header with the grouping column value, you can simply put the grouping column on the right side of the data:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	softartisans													
2	Northeast Subscriptions Tracking Report													
3	DIVISION - ALL													
4		February , 2008												
5		New Subscriptions				Subscription Renewals				Total				
6		#	Revenue	Avg. Price	%	#	Revenue	Avg. Price	%	#	Revenue	Avg. Price		
7													%%group	
8													%%header	
9	%%value(Data.Region)													
10	%%=Data.Game	%%=Data	%%=Data	#VALUE!	0%	%%=Data	%%=Data	#VALUE!	0%	0	$0	$0	%%=Data.Region(hide)	
11													%%footer	
12	Total	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0		
13													%%endgroup	
14	TOTAL SUMMARY BY GAME													
15	Legend of Fred	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0		
16	Generic Spy Shooter	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0		
17	Capture the Flag	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0		
18	Legacy Fantasy MMORPG	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0		
19	Legacy Sci-Fi MMORPG	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0		
20	Legacy RTS	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0		
21	GRAND TOTAL AMOUNT	0	$0	$0	0%	0	$0	$0	0%	0	$0	$0		

The grouping markers will be removed, and the **(hide)** data marker modifier tells ExcelTemplate to remove the grouping column data as well, so that we do not repeat the values:



	A	B	C	D	E	F	G	H	I	J	K	L
1	softartisans											
2	Northeast Subscriptions Tracking Report											
3	DIVISION - ALL											
4		February , 2008										
5		New Subscriptions				Subscription Renewals				Total		
6		#	Revenue	Avg. Price	%	#	Revenue	Avg. Price	%	#	Revenue	Avg. Price
7	Rochester											
8	Legend of Fred	111	$5,772	$52	30%	259	$5,180	$20	70%	370	$10,952	$30
9	Generic Spy Shooter	284	$12,780	$45	100%	0	$0	$0	0%	284	$12,780	$45
10	Capture the Flag	10	$400	$40	63%	6	$120	$20	38%	16	$520	$33
11	Legacy RTS	0	$0	$0	0%	99	$1,980	$20	100%	99	$1,980	$20
12	Total	405	$18,952	$47	53%	364	$7,280	$20	47%	769	$26,232	$34
13	New York											
14	Legend of Fred	116	$6,148	$53	89%	15	$300	$20	11%	131	$6,448	$49
15	Generic Spy Shooter	1548	$71,208	$46	100%	0	$0	$0	0%	1548	$71,208	$46
16	Capture the Flag	3	$126	$42	33%	6	$120	$20	67%	9	$246	$27
17	Legacy Fantasy MMORPG	0	$0	$0	0%	1	$20	$20	100%	1	$20	$20
18	Legacy Sci-Fi MMORPG	0	$0	$0	0%	1	$20	$20	100%	1	$20	$20
19	Legacy RTS	0	$0	$0	0%	129	$2,580	$20	100%	129	$2,580	$20
20	Total	1667	$77,482	$46	92%	152	$3,040	$20	8%	1819	$80,522	$44
21	Buffalo											
22	Legend of Fred	309	$16,068	$52	55%	250	$5,000	$20	45%	559	$21,068	$38
23	Generic Spy Shooter	337	$15,839	$47	97%	10	$200	$20	3%	347	$16,039	$46
24	Capture the Flag	21	$903	$43	70%	9	$180	$20	30%	30	$1,083	$36
25	Legacy Fantasy MMORPG	0	$0	$0	0%	5	$100	$20	100%	5	$100	$20

ExcelTemplate Output Options

After creating a workbook with ExcelTemplate, you can:

- Save the workbook to disk.
- Write the workbook to a stream.
- Stream the workbook to the client and open it in Excel.
- Stream the workbook to the client and open it in the browser.
- Pass the workbook to ExcelApplication.

Save(outputFileName)

This method saves the generated Excel file on the server.

outputFileName specifies a complete path and file name for the generated file. ExcelWriter will save the file to this location. If a file with the same name exists, it will be overwritten by the new Excel file.

Example

C#

```
excelTmpl.Save(@"C:\Reports\Orders.xls");
```

VB.NET

```
excelTmpl.Save("c:\Reports\Orders.xls")
```

Save(outputStream)

This method writes the generated Excel file to the specified System.IO.Stream, or a class derived from System.IO.Stream (for example, System.IO.FileStream).

If you pass Save a System.IO.FileStream, ExcelWriter will save the generated file on the server. If you pass Save Response.OutputStream, ExcelWriter will stream the the generated file to the client.

Example

C#

```
//-- FileStream and FileMode are in the System.IO namespace
FileStream fstream = new FileStream(@"C:\temp\outfile.xls", FileMode.Create);

//-- Pass the FileStream to ExcelTemplate
excelTmpl.Save(fstream);

//-- Close the FileStream (could be in a finally block)
fstream.Close();
```

VB.NET

```
'-- FileStream and FileMode are in the System.IO namespace
Dim fstream As New FileStream("C:\temp\outfile.xls", FileMode.Create)

'-- Pass the FileStream to ExcelTemplate
excelTmpl.Save(fstream)

'-- Close the FileStream (could be in a finally block)
fstream.Close()
```

Save(response)

This method streams the generated Excel file to the client.

If you pass Save an HttpResponse object, ExcelWriter 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. The browser will use the ExcelWriter script name as the default file name displayed in the File Download dialog. To set a different file name and/or to open the file in Microsoft Excel, use the signature Save(response, attachmentName, OpenInBrowser).

Example

C#

```
excelTmpl.Save(Page.Response);
```

VB.NET

```
excelTmpl.Save(Page.Response)
```

Save(response, attachmentName, OpenInBrowser)

This method streams the generated Excel file to the client.
If you pass Save an HttpResponse object, ExcelWriter 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 Excel.

The parameter attachmentName specifies a name for the generated Excel file; this name will be displayed in the download dialog when the file is streamed to the browser. If the parameter 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 Excel. By default, the file will open in the browser window. Note: not all browsers can embed an Excel file in the browser window.

C#

```
excelTmpl.Save(Page.Response, "Output.xls", false);
```

VB.NET

```
excelTmpl.Save(Page.Response, "Output.xls", False)
```

Pass the Template to ExcelApplication

> Though the ExcelTemplate object supports Excel's BIFF8 (Excel 97/2000/XP/2003) and Office Open XML (Excel 2007) formats, the ExcelApplication object does not yet support Office Open XML (Excel 2007) formats. Templates based on Office Open XML files (.xlsx and .xlsm) should not be passed to the ExcelApplication object.

You can use ExcelTemplate to open and populate an ExcelWriter template, then pass the populated workbook to ExcelApplication for additional processing. In this case, do not call Save. Instead, pass the ExcelTemplate object to ExcelApplication's open method:

C#

```
ExampleExcelTemplate xlt = new ExcelTemplate();

xlt.Process();

ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Open(xlt);
```

Passing ExcelTemplate to ExcelApplication

ExcelWriter allows you to generate a spreadsheet from script alone - using the ExcelApplication object - or from a template spreadsheet and a script, using ExcelTemplate. ExcelTemplate provides an intuitive high-performance way to import database values to a spreadsheet, but cannot otherwise modify a spreadsheet at runtime. ExcelApplication's rich object model allows you to modify every aspect of the spreadsheet at runtime. You can take advantage of the features of both ExcelApplication and ExcelTemplate by using them together. For example, you can use ExcelTemplate to open and populate an ExcelWriter template, then pass the populated workbook to ExcelApplication and add a chart.

> Though the ExcelTemplate object supports Excel's BIFF8 (Excel 97/2000/XP/2003) and Office Open XML (Excel 2007) formats, the ExcelApplication does not yet support Office Open XML (Excel 2007) formats. Templates based on Office Open XML files (.xlsx and .xlsm) should not be passed to the ExcelApplication object.

To pass a workbook from ExcelTemplate to ExcelApplication, do not call ExcelTemplate.Save. Instead, pass the ExcelTemplate object to ExcelApplication's Open method:

```
ExcelTemplate xlt = new ExcelTemplate();
xlt.Open(templatePath);
xlt.BindData(data, "Sales", xlt.CreateDataBindingProperties());
xlt.Process();

//--- Create an instance of ExcelApplication and
//--- open the spreadsheet you created with ExcelTemplate.
//--- The spreadsheet will be returned as a Workbook
//--- object.
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Open(xlt);
```

Code Sample: Passing ExcelTemplate to ExcelApplication

[C#] | [VB.NET]

Templates and Pivot Tables

> The instructions on this page apply for using pivot tables with **ExcelWriter 7.6 and above**. If you are using ExcelWriter 7.5 or earlier please use the instructions on Templates and Pivot Tables in Older Versions

A PivotTable report is an interactive table which allows the user to have multiple views of data. By changing the view, the PivotTable report can show different summaries of the data or drill down to display more details for particular areas. ExcelWriter allows you to include PivotTables in an ExcelWriter template file. If there are PivotTables in the original template file, they will be preserved when the template is processed using ExcelWriter.

> When using the ExcelApplication object to modify an existing spreadsheet, the following steps are not required; these steps are necessary only when opening a file containing data markers (i.e. an ExcelWriter template).

- What is a data marker?
- Data markers and PivotTables
- How to Create an ExcelWriter Template with PivotTables

What is a data marker?

An ExcelWriter **template** is a file created in Microsoft Excel that contains data markers. A **data marker** is a cell value beginning with **%%=** or **%%=$** that specifies a database column, variable, or array to insert in the spreadsheet cell or column containing the marker. ExcelWriter gets values at runtime from a data source and imports them to spreadsheet data markers.

For detailed information on data markers, see Creating Data Markers.

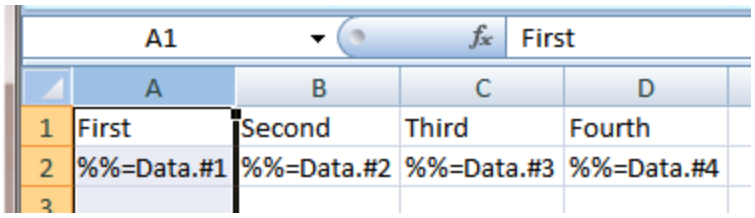
Data markers and PivotTables

ExcelTemplate data markers can be used to bind data to a template for use with a Pivot Table. When ExcelTemplate binds the data to the data markers, it also updates the range that the PivotTable uses as a pivot table cache. By setting the pivot tables to refresh on opening, they will automatically bring in the new values from the pivot table cache.

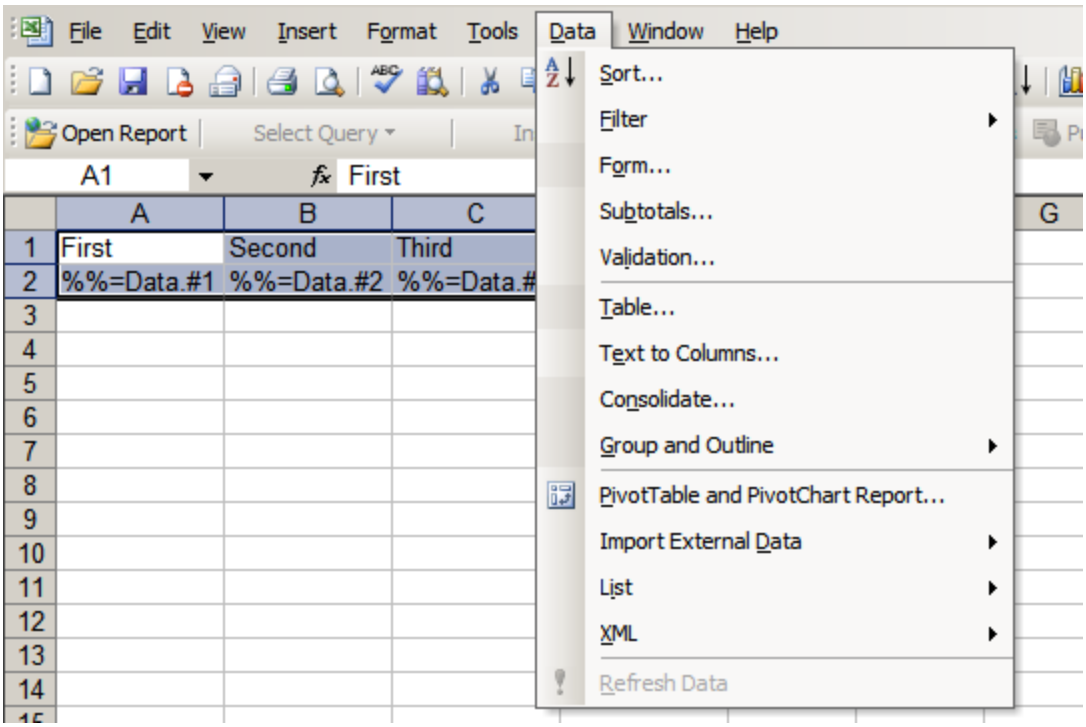
How to Create an ExcelWriter Template with PivotTables

1. Create a template file containing data markers, each with their own column heading (see How to Use Templates). **Do not create any of**

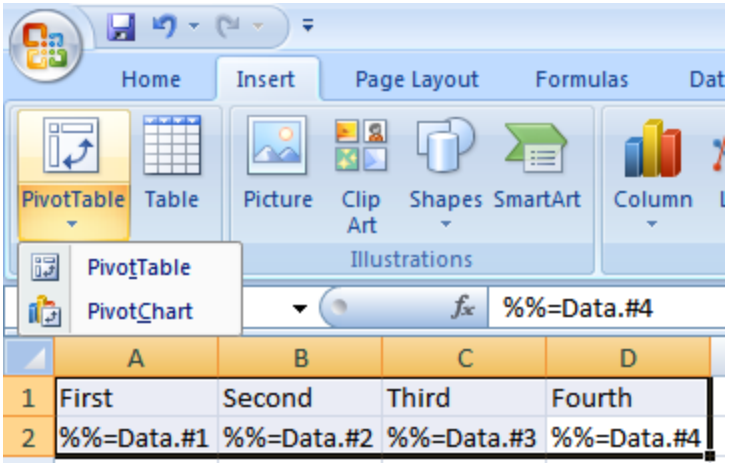
the PivotTables yet.



2. **In Excel 2003:** Use Microsoft Excel's *PivotTable and PivotChart Report Wizard* to create a PivotTable referencing the range of cells containing the column headings and data markers.

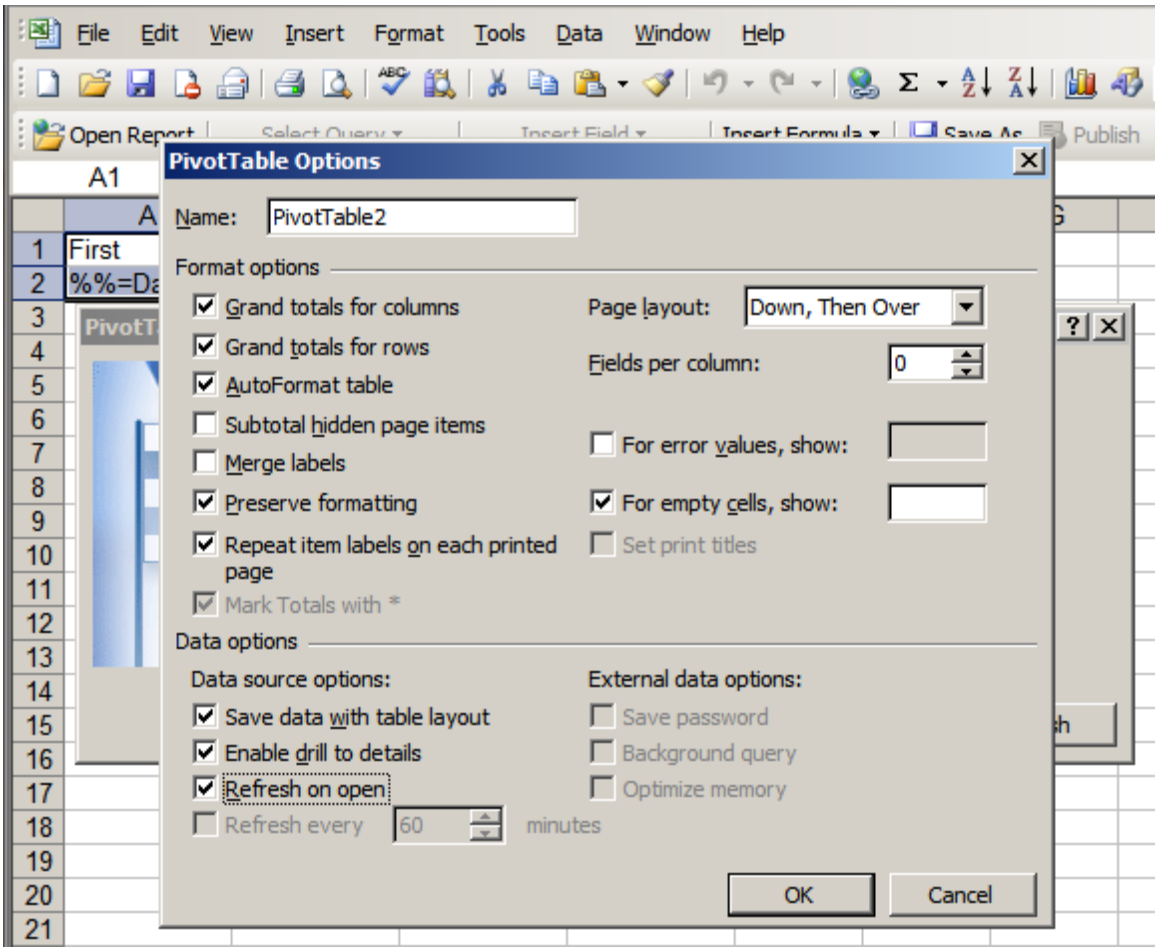


In Excel 2007 or above: Use the Ribbon controls (in Excel 2007 and beyond) to create a PivotTable referencing the range of cells containing the column headings and data markers.

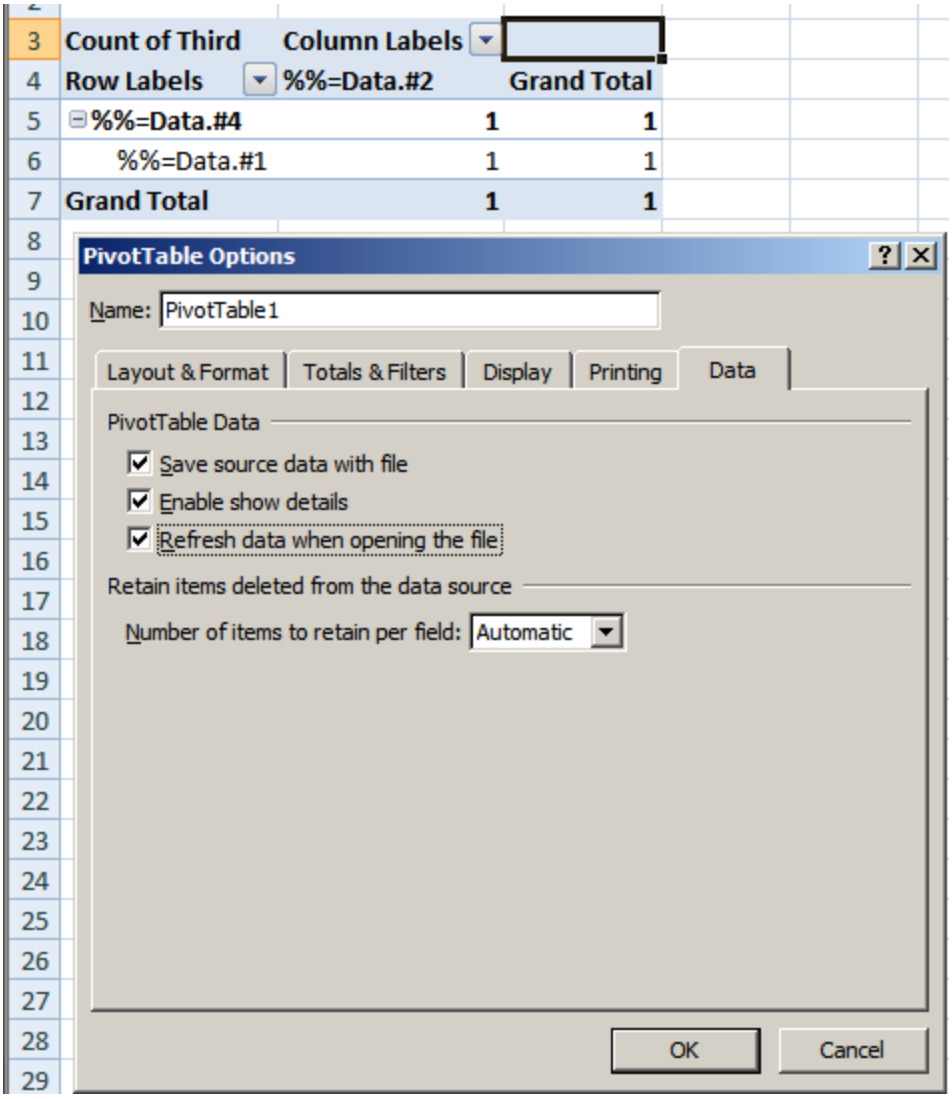


3. Specify the initial layout and field formatting for the PivotTable. Place the PivotTable in any worksheet you desire. (See the Excel documentation for more details about creating your PivotTable.)

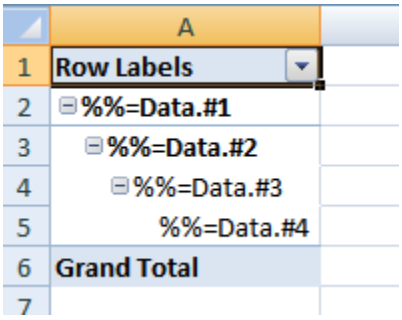
4. **In Excel 2003:** Before clicking "Finish" in the Wizard, click the "Options" button and make sure that "Refresh on open" is **checked**.



In Excel 2007 or above: After creating the Pivot Table, right-click on the table and select "PivotTable Options". Click on the Data tab and make sure that "Refresh data on opening the file" is **checked**.



5. In your newly created PivotTable, you will now see the data markers. Data markers in the pivot table will be ignored by ExcelWriter, which will instead update the data source.



6. Repeat steps 2 through 5 for every PivotTable you wish to create. Finish creating your template file and save it in the location on your server which will be accessed by your ExcelWriter script.

Templates and Pivot Tables in Older Versions

> The instructions on this page apply for using pivot tables with **ExcelWriter 7.5 and earlier**. If you are using ExcelWriter 7.6 or later please use the instructions on Templates and Pivot Tables

A PivotTable report is an interactive table which allows the user to have multiple views of data. By changing the view, the PivotTable report can

show different summaries of the data or drill down to display more details for particular areas. ExcelWriter allows you to include PivotTables in an ExcelWriter template file. If there are PivotTables in the original template file, they will be preserved when the template is processed using ExcelWriter.

There are specific guidelines you must follow when creating a template file for bringing data into PivotTables using ExcelTemplate.

> When using the ExcelApplication object to modify an existing spreadsheet, the following steps are not required; these steps are necessary only when opening a file containing data markers (i.e. an ExcelWriter template).

- What is a data marker?
- Data markers and PivotTables
- How to Create an ExcelWriter Template with PivotTables

What is a data marker?

An ExcelWriter **template** is a file created in Microsoft Excel that contains data markers. A **data marker** is a cell value beginning with **%%=** or **%%=$** that specifies a database column, variable, or array to insert in the spreadsheet cell or column containing the marker. ExcelWriter gets values at runtime from a data source and imports them to spreadsheet data markers.

For detailed information on data markers, see Creating Data Markers.

Data markers and PivotTables

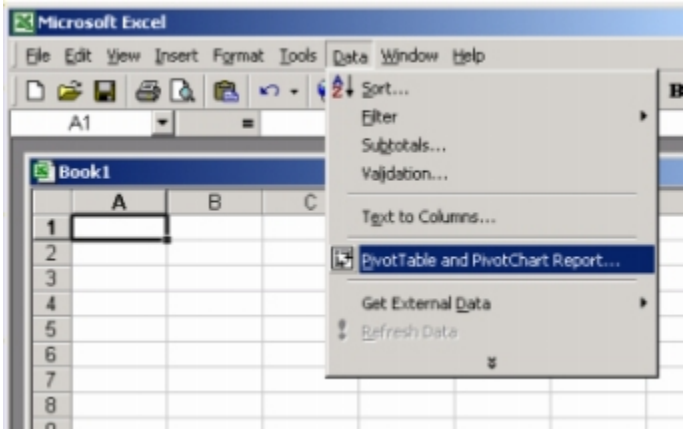
ExcelTemplate data markers can be used to bind data to a template for use with a Pivot Table. However, if you add a data marker row as part of a pivot table data source without taking any further steps, the data markers will be pulled into the Pivot Table cells. When ExcelTemplate binds data to the template, it will see these data markers (as well as subtotal cells) as regular data markers and attempt to process them, resulting in an error.

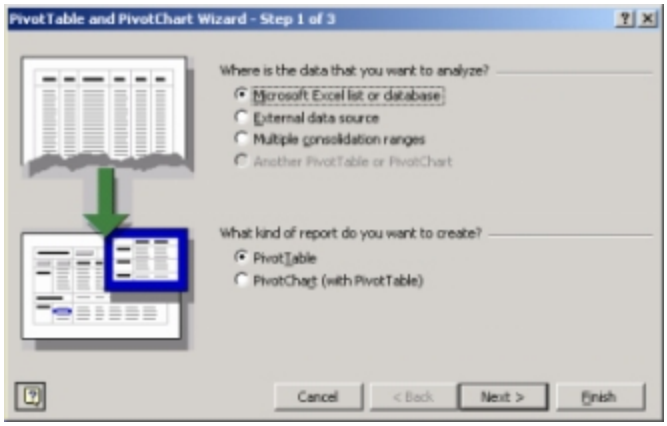
To solve this problem, you will need to do two things:

- Replace the data markers in the PivotTables with temporary data.
- Keep 2 copies of your template file, one for editing and one for production. In the editable version, the PivotTables will contain temporary data and should never be refreshed. In the production version, the PivotTables must be set to "Refresh on open" so that the newly-loaded data will populate the PivotTables of the ExcelWriter-generated spreadsheet.

How to Create an ExcelWriter Template with PivotTables

1. Create a template file containing data markers, each with their own column heading (see How to Use Templates). **Do not create any of the PivotTables yet**.





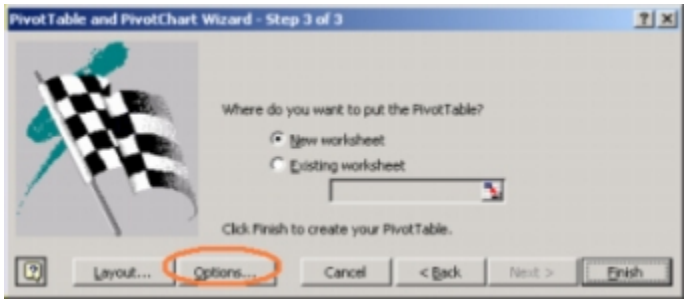
2. Copy the data markers row to another location on the worksheet. Replace the original data markers with arbitrary values of the same data type, which we call "dummy data".

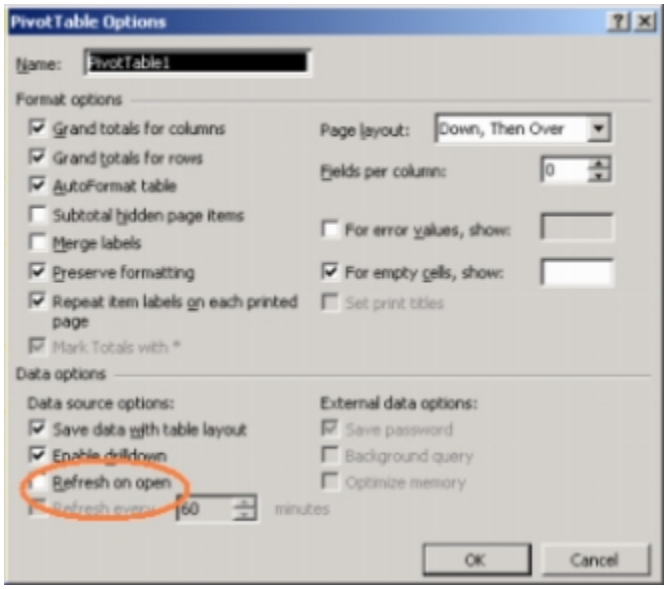
3. **In Excel 2003:** Use Microsoft Excel's *PivotTable and PivotChart Report Wizard* to create a PivotTable referencing the range of cells containing the column headings and dummy data.

   In Excel 2007 or above: Use the Ribbon controls (in Excel 2007 and beyond) to create a PivotTable referencing the range of cells containing the column headings and dummy data.

4. Specify the initial layout and field formatting for the PivotTable. Place the PivotTable in any worksheet you desire. (See the Excel documentation for more details about creating your PivotTable.)

5. **In Excel 2003:** Before clicking "Finish" in the Wizard, click the "Options" button and make sure that "Refresh on open" is **not checked**.





   In Excel 2007 or above: After creating the Pivot Table, right-click on the table and select "PivotTable Options". Click on the Data tab and make sure that "Refresh on open" is **not checked**.

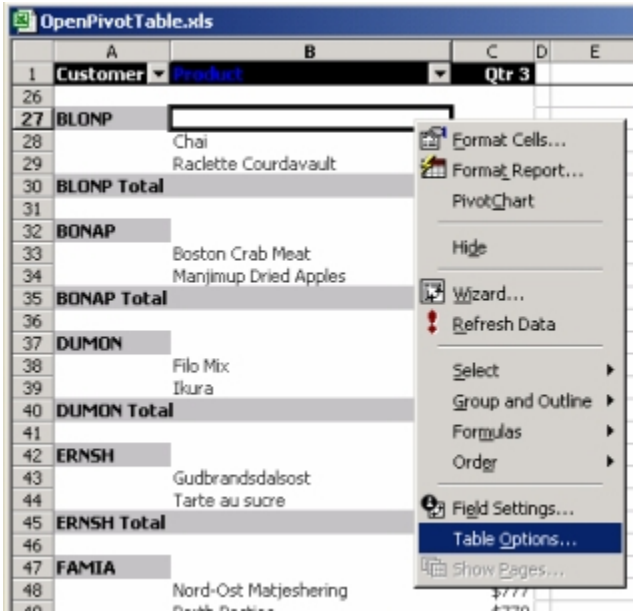
6. In your newly created PivotTable, you will now see the dummy data. **Replace** the dummy data on the data sheet with the original data markers.

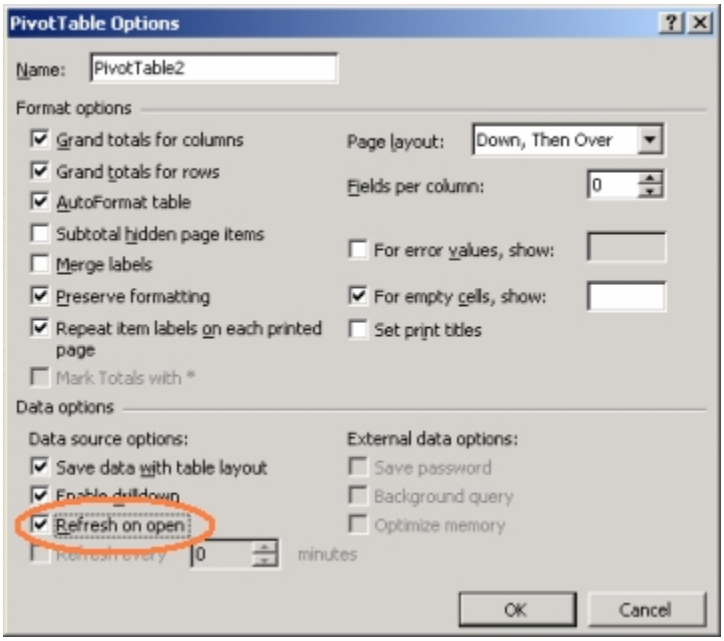
7. Repeat steps 2 through 6 for every PivotTable you wish to create. Finish creating your template file and save it as a version which you may edit in the future, for example "template_edit.xls".

8. Now, save another copy of the template file for production, for example "template_production.xls"

9. Set the pivot table to refresh on open using the following methods:

   In Excel 2003: Right-click on a PivotTable and select Table Options. **Check** "Refresh on open". **Repeat this for every PivotTable in your workbook**. You need to enable refreshing in the production copy in order for the newly-loaded data to automatically fill your PivotTables when the ExcelWriter-generated workbook is opened on the client.





   In Excel 2007 or above: Right-click on a PivotTable and select PivotTable Options. Select the "Data" tab and **Check** "Refresh on open". **Repeat this for every PivotTable in your workbook**. You need to enable refreshing in the production copy in order for the newly-loaded data to automatically fill your PivotTables when the ExcelWriter-generated workbook is opened on the client.

10. Save and close the production copy of your workbook and place it in the location on your server which will be accessed by your ExcelWriter script. **Never open this file in Excel**. If you open the production copy of the template in Excel, the tables will refresh, pulling in the data markers.

11. If you wish to make changes in your template file, open your editable version. Again, **do not** refresh the tables in the editable version. After your changes are complete, repeat steps 6 through 9 and overwrite the existing production version.

Stretching Formulas

A data marker is a single cell value. But a data marker will usually be populated with several rows of cell values. For example, if you set the data source for the data marker %%=Products.ProductID, to the "ProductID" column in a database table, the marker cell and the cells below it will be populated with ProductID values. New rows will be inserted between the data marker row and the next row in the template that contains a value. This can affect formulas because the range of cell values a formula should reference may change when the workbook is populated.

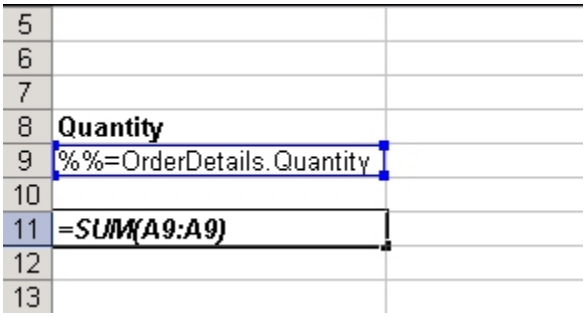
If, for example, template cell A11 contains the formula =SUM(A9:A9), where A9 contains the marker %%=OrderDetails.Quantity, the range of values that should be added will change after a column of values is inserted at cell A11. In this case, ExcelWriter will stretch the formula so that it references the range of cells from A9 to the last cell that contains an imported value. And the original formula will be pushed down because new rows were inserted between it and the data marker row.

ExcelWriter will move or modify a formula depending on where it is in the template, and whether it references data marker cells:

- If a formula references cells **above** all data marker rows, ExcelWriter will not move or modify it.
- If a formula references cells **below** a data marker row, the references will be updated if new rows are inserted between the data marker row and the cells referenced in the formula.
- If a formula references a data marker cell, the formula will be "stretched" to include the imported cell values.

Example

In the ExcelWriter template, cell A11 contains the formula **=SUM(A9:A9)**.



In the populated workbook, the formula is in cell A28 and calculates the sum of values A9 to A26.

6			
7			
8	**Quantity**		
9	12		
10	10		
11	5		
12	9		
13	40		
14	10		
15	35		
16	15		
17	6		
18	15		
19	20		
20	40		
21	25		
22	40		
23	20		
24	42		
25	40		
26	15		
27			
28	***399***		
29			

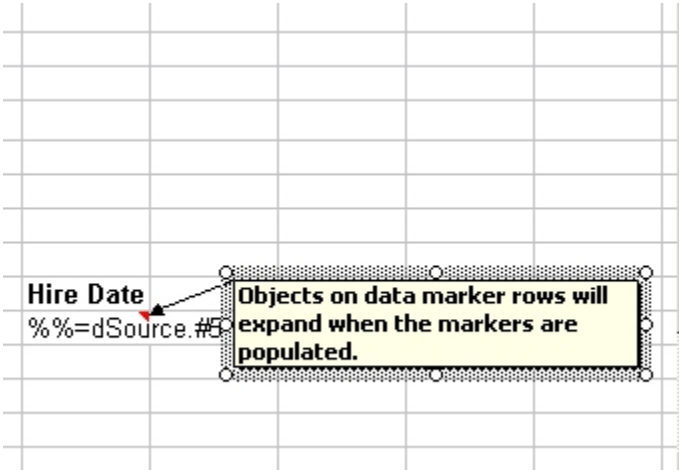
Drawing Objects and Data Markers

If you include drawing objects (charts, comments, etc.) in a template, ExcelWriter will preserve the objects in the generated workbook. However, depending on where the object is in the template, ExcelWriter may move or modify it:

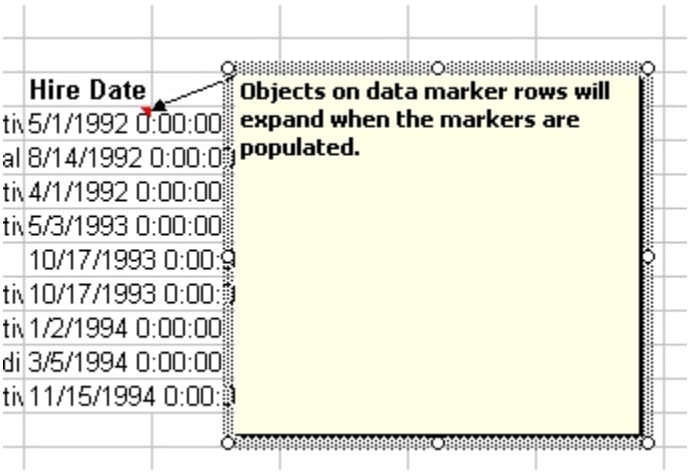
- If an object is **above** all data marker rows, ExcelWriter will not move or modify it.
- If an object is **below** a data marker row, it will be pushed down if new rows are inserted between the object and the data marker when the marker is populated.
- If an object is **on** a data marker row, and a column of values is imported to the marker from the data source, the object will be expanded downward so that it is long as the column of imported values.

Example

A comment on a template data marker row:



The same comment after the data marker is populated:



ExcelApplication in Depth

- Addressing Cells
- Formulas
- Styles in ExcelWriter
- Number Formats
- The Color Palette
- Areas and Ranges
- Modifying an Existing Spreadsheet
- Importing Data
- Charts in ExcelApplication
- Drawing Objects
- Output Options for ExcelApplication
- Passing ExcelApplication to ExcelTemplate
- Formatting Headers and Footers
- Protecting Your Worksheet
- Copying Data From a Worksheet
- Multilingual Support

Addressing Cells

There are three ways to address cells, areas, and ranges using ExcelApplication:

- Excel's A1 Reference Style
- Excel's R1C1 Reference Style
- Row Number and Column Number

Excel's A1 Reference Style

A1-style references are easy to read, intuitive, and match the default reference style seen in Microsoft Excel.

Example

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheets[0];
ws.Cells["A3"].Value = "Jan";
ws.Cells["B3"].Value = "Feb";
ws.Cells["C3"].Value = "Mar";
Area a = ws.CreateArea("=A1:G10");
```

A1 cell references are relative by default. Therefore, if you enter "=B1" in cell A1 and drag A1 down, the formula in cell Ax will be "=Bx". To make an A1 row or column reference absolute, prepend a dollar sign to the row or column. For example, if you enter "=B$1" in cell A1 and drag A1 down, the formula in cell Ax will be "=B$1".

The reference style used in the generated workbook (when it is opened in Excel) is determined by the value of the property Workbook.UseRCFormulaNotation. If it is false (the default value), Excel will use A1 references. If UseRCFormulaNotation is set to true, Excel will use R1C1-style references.

Excel's R1C1 Reference Style

In the R1C1 style, the location of a cell is specified with an "R" followed by a row number and a "C" followed by a column number. R1C1 references can be either absolute or relative. To create an absolute reference, specify the row and column numbers without brackets. For example, "=R1C2" equates to row 1, column 2. To create a relative reference for a cell, specify the row and column numbers in brackets; and enter the difference between that cell's row and column numbers and those of the cell it references. Use negative numbers to reference cells above or to the left of a cell. Use positive numbers to reference cells below or to the right of a cell. For example, entering "=R[-2]C[-1]" in cell R3C2 causes it to refence the value of cell R1C1.

Example

Relative R1C1 references enable you to reuse a single formula in multiple rows or columns. The following example copies one formula to three different columns to calculate the sum of those columns.

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
wb.UseRCFormulaNotation = true;
Worksheet ws = wb.Worksheets[0];
String formulaString = "=SUM(R[-12]C:R[-1]C)";
Cell columnTotal;
for (int i = 0;i < 3; i++)
{
     columnTotal = ws.Cells[13, i + 1];
     columnTotal.Formula = formulaString;
}
```

The reference style used in the generated workbook (when it is opened in Excel) is determined by the value of the property Workbook.UseRCFormulaNotation. If it is false (the default value), Excel will use A1 references. If UseRCFormulaNotation is set to true, Excel will use R1C1-style references.

There are several valid ways of addressing a cell or an area using RC notation:

Single Cell, Absolute	RxCy
Single Cell, Row Relative	R[x]Cy
Single Cell, Column Relative	RxC[y]
Single Cell, Both Relative	R[x]C[y]
Single Cell, Same Relative Row, Absolute Column	RCy
Single Cell, Same Relative Row, Relative Column	RC[y]
Single Cell, Absolute Row, Same Relative Column	RxC
Single Cell, Relative Row, Same Relative Column	R[x]C
Entire Row, Absolute	Rx
Entire Row, Relative	R[x]
Entire Row, Same, Relative	R
Entire Column, Absolute	Cy
Entire Column, Relative	C[y]
Entire Column, Same, Relative	C

Row Number and Column Number

Addressing cells by number allows you to iterate over cells in your ExcelWriter code. Row and Column number references are also processed faster than Excel-style references.

The following specifies cell A1 by 0-based row and column numbers: The first parameter of the Cell property is the row index, the second is the column index.

```
sheet1.Cells[2, 0].Value = "SoftArtisans OfficeWriter";
```

Using row and column numbers makes it easy to iterate over cells:for(int iRow = 3; iRow <= 23; iRow++)

```
for(int iCol = 0; iCol <= 2; iCol++)
     sheet1.Cells[iRow, iCol].Value = iRow + iCol;
```

Formulas

ExcelApplication supports all math, string, boolean, time, statistical, database, lookup and reference formulas or functions that are part of Excel. ExcelWriter does not calculate formulas at run-time; all formulas are calculated when the workbook is opened in Excel.

To insert a formula in a cell, use the propertyCell.Formula. Begin the formula string with =, as you would when entering a formula in Microsoft Excel:

```
sheet1.Cells["A25"].Formula = "=SUM(A4:A24)";
```

ExcelWriter allows you to use values from a different sheet when assigning a formula to a cell, as in the following examples:

```
sheet1.Cells[5,3].Formula = "=SUM(Sheet1!A1:A3,Sheet3!A1:A5)";
```

The property Cell.Name returns a cell's Excel-style name (e.g., "B5"). This property can be useful if you want to build a formula string using row and column indexes, rather than cell names, for example:

```
String formula = "=SUM(" + sheet1.Cells[3, 1].Name +
     ":" + sheet1.Cells[23, 1].Name + ")";
sheet1.Cells[24, 1].Formula = formula;
```

Example

Here is an example ASP.NET Page class that demonstrates how to use Formulas with ExcelApplication:

```
using SoftArtisans.OfficeWriter.ExcelWriter;

public class FormulaDemo : System.Web.UI.Page
{
     protected void Page_Load(object sender, System.EventArgs e)
     {
          //--- Create ExcelApplication, a Workbook, and a Worksheet
          ExcelApplication xlw = new ExcelApplication();
          Workbook wb = xlw.Create();
          Worksheet sheet1 = wb.Worksheets[0];

          //--- Write some values into cells for the
          //--- formulas to compute
          sheet1.Cells["A1"].Value = 1;
          sheet1.Cells["A2"].Value = 2;
          sheet1.Cells["A3"].Value = 3;

          //--- Write a formula into a cell
          //--- Formula will be calculated
          //--- when opened in Excel
          sheet1.Cells["A4"].Formula = "=SUM(A1:A3)";

          //--- Save the workbook
          xlw.Save(wb, Page.Response, "Generated.xls", false);
     }
}
```

Code Sample: ExcelApplication Basic Steps

This sample shows many of the core features of ExcelApplication, including the use of formulas.

[C#] | [VB.NET]

Styles in ExcelWriter

ExcelWriter creates three style types:

- GlobalStyle
- NamedStyle
- CellStyle

All three style types derive from the Style class, and NamedStyle derives from GlobalStyle.

A style can be *set* or *applied* to cells, rows, columns, ranges, and areas.

When a style is *set*, the object to which the style is assigned acquires all of that style's properties, including font properties and number formatting. When a style is *applied*, only the differences between the new style and existing style properties (assigned through the ExcelWriter API or in Microsoft Excel) will take effect. For example, if the cell has a background color and the new style applied does not contain a background color, the cell's color will not be affected. Note: Properties that were set on a pre-existing style in a workbook will not be propogated to the object to which a new style is *applied*.

> ⚠ Setting a style overwrites any pre-existing formatting on the cell, area, or range. Applying a style merges the new style with the previous one.

To *set* a style, use the Style property or the SetStyle method. To *apply* a style, call ApplyStyle. Both are accessible through the following objects: Cell, Area, Range, RowProperties, and ColumnProperties.

GlobalStyle

A GlobalStyle is global to a workbook. It can be *set* or *applied* to cells, rows, columns, areas, and ranges. When a GlobalStyle is *set* (using the Style property), subsequent changes to the style affect all cells on which the style was set. When a GlobalStyle is *applied* (using the ApplyStyle method), subsequent changes to the style will not affect cells to which the style was applied.

Unlike NamedStyles, GlobalStyles are not stored in a collection, so they may be created, but not retrieved. A GlobalStyle is not accessible after

the workbook is saved.

To create a GlobalStyle, call Workbook.CreateStyle.

NamedStyle

The ExcelApplication object contains a collection of NamedStyles, which - unlike GlobalStyles are accessible after the workbook is saved. This collection includes Excel's built-in styles and any user-defined styles in a workbook opened with the ExcelApplication.Open method.

A NamedStyle is global to a workbook. It can be *set* or *applied* to cells, rows, columns, areas, and ranges. When a NamedStyle is *set* (using the Style property), subsequent changes to the style affect all cells on which the style was set. When a NamedStyle is retrieved from an existing workbook and *applied* (using the ApplyStyle method), subsequent changes to the style will not affect cells to which the style was applied.

To create a NamedStyle, call Workbook.CreateNamedStyle.

To return an existing NamedStyle, call Workbook.GetNamedStyle.

CellStyle

Every cell in an Excel workbook contains a unique cell style. This style is exposed by ExcelWriter through the CellStyle class. Changes to a CellStyle affect only the cell that owns the style. A CellStyle may be *set* on another cell or group of cells, but this action clones the style. Changes to the original CellStyle reference affect only the owning cell and changes to the cloned CellStyle affect only the cell on which the style was set. If the style is set on a cell grouping object - such as an area, range, row, or column - each cell in the grouping receives a unique clone of the style.

CellStyles are accessed through Cell.Style.

Example

```
using SoftArtisans.OfficeWriter.ExcelWriter;

class StylesDemo : System.Web.UI.Page
{
     protected void Page_Load(object sender, System.EventArgs e)
     {
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];

          //--- Write some random data into the cells to be stylized
          System.Random rand = new System.Random();
          for(int iRow = 0; iRow < 24; iRow++)
               for(int iCol = 0; iCol < 3; iCol++)
                    ws.Cells[iRow, iCol].Value = rand.Next(100);

                    ws.Cells[24, 0].Formula = "=SUM(A1:A24)";
                    ws.Cells[24, 1].Formula = "=SUM(B1:B24)";
                    ws.Cells[24, 2].Formula = "=SUM(C1:C24)";

                    //--- This GlobalStyle, styleMoneyFormat, will be used to apply currency
                    //--- stylization to workbook values.
                    Style styleMoneyFormat = wb.CreateStyle();
                    styleMoneyFormat.NumberFormat = "$#,##.00";

                    //--- styleBold represents a bold-faced font.
                    Style styleBold = wb.CreateStyle();
                    styleBold.Font.Bold = true;

                    //--- Set Style on an Area of cells
                    //--- The SetStyle method sets a base style
                    //--- and overwrites all existing style attributes
                    Area areaData = ws.CreateArea("A1:C25");
                    areaData.SetStyle(styleMoneyFormat);

                    //--- ApplyStyle overlays a style with an existing
                    //--- style, changing only the attributes that have
                    //--- been defined in the applied style
                    Area areaTotalRow = ws.CreateArea("A25:C25");
                    areaTotalRow.ApplyStyle(styleBold);

                    xla.Save(wb, Page.Response, "Styles.xls", false);
               }
          }
     }
}
```

Code Eample: ExcelApplication Basic Steps

This sample shows many of the core features of ExcelApplication, including the use of Styles.

[C#] | [VB.NET]

Number Formats

A number format is a string that is used to format dates or numbers. For example, if you enter the value 0.563 in cell A4 and the number format string assigned to cell A4 is "0.00%", the value displayed will be 56.30%. Changing a cell's number format will not change the actual value that is used in calculations. In Microsoft Excel, you will see the cell's actual value in the formula bar.

Number Formats in Excel

In Excel, number formats can be assigned to cells through the **Format Cells** dialog. To apply a new number format to cells in Excel:

1. Select a cell or group of cells.
2. From the **Format** menu, select **Cells**, or, right-click the highlighted cells and select **Format Cells**.
3. Select the **Number** tab.

4. Select a number format category from the list on the left.
5. Set additional attributes (if available) on the right and click **Ok**.

How to Set Number Formats with ExcelWriter

To assign a number format with ExcelWriter's ExcelApplication, use the Style object's NumberFormat property. The following lines create a percentage style and assign it to a cell:

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();

//--- Create a percentage style and assign
//--- a number format to it.
GlobalStyle stylPercent = wb.CreateStyle();
stylPercent.NumberFormat = "0.00%";

//--- Assign a numeric value and
//--- stylPercent to cell A4 to display
//--- 56.30% in the cell.
ws.Cells["A4"].Value = "0.563";
ws.Cells["A4"].Style = stylPercent;
```

ExcelWriter includes a set of built-in format constants to help you quickly assign commonly used display formats. For example, the constant NumberFormat.DateFormat.DayOfWeekMonthDayYear represents the format string "dddd, mmmm dd, yyyy". The following assigns this format to a date style:

```
GlobalStyle styleDate = wb.CreateStyle();
styleDate.NumberFormat = NumberFormat.DateFormat.DayOfWeekMonthDayYear;
```

For a complete list of number format constants, see The NumberFormat Object.

The NumberFormat Object

The NumberFormat object is a helper class that you can use to create custom number format strings. NumberFormat contains methods for creating accounting, currency, fraction, number, percentage, and scientifice display formats. For example, the method NumberFormat.CreateAccounting creates an accounting format string. This method takes three parameters:

numDecimalPlaces	The number of decimal places to display.
useDollarSign	If true, the dollar sign will be displayed.
negativeColor	A color to use when displaying negative numbers. This parameter may be null.

The following example creates an accounting style with two decimal places, a dollar sign and negative values in red:

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();

GlobalStyle stylCurrency = wb.CreateStyle();
NumberFormat numFormat = wb.NumberFormat;
String acctFormat = numFormat.CreateAccounting(2, true, NumberFormat.Color.Red);
stylCurrency.NumberFormat = acctFormat;

Area ar = wb.Worksheets[0].CreateArea("B7:H24");
ar.ApplyStyle(stylCurrency);
```

The Color Palette

Each Excel document has a single color palette that manages formatting colors and some drawing layer colors. A color palette contains 56 colors in Excel binary (.xls) format. Each of these colors may be customized, but changing a color in the palette will affect all formatting that uses the referenced color.

When ExcelWriter opens a workbook that does not contain an explicitly defined palette, or when ExcelWriter creates a new workbook, it uses the same default color palette that Excel uses. ExcelWriter never automatically adds colors to the palette.

Workbook.Palette returns a Palette object that represents the workbook's color palette. A Color object represents a single color in the palette.

To replace a color in the palette with a new color, call either of the following methods:

- `Palette.SetColor(Color color, int red, int green, int blue)`

  This method takes the color to replace as a Color object, and the RGB values of the new color.

- `Palette.SetColorAt(int index, int red, int green, int blue)`

  This method takes the 0-based index of the color to replace, and the RGB values of the new color.

If you copy an object or cells with a custom color to another workbook, the custom color is replaced by the color in the corresponding position on the other workbook's color palette. To retain the custom color, either copy the customized color palette to the other workbook or change the corresponding color in the workbook.

If you are not sure that a specific color exists in the palette, and you want to assign the color to an element in the workbook, you can call Palette.GetClosestColor. Pass the method the red, green, and blue values of the ideal color. Palette.GetColor retrieves the color with the exact RGB value specified; if this color is not in the palette, an exception will be thrown.

A workbook's palette contains ten system colors that are fixed and cannot be changed by the user: SystemAutomaticFill, SystemAutomaticLine SystemBlack, SystemBlue, SystemCyan, SystemGreen, SystemMagenta, SystemRed, SystemWhite, and SystemYellow.

Drawing objects (pictures, shapes, comments, etc.) support custom RGB colors that may or may not be present in the palette. If you modify the color of an AutoShape (right-click the shape and select "Format AutoShape"), for example, and select a non-palette custom color, the change will not affect the workbook palette or other elements in the workbook to which the replaced color was assigned. To set custom values for shape fill and line colors, use Shape.SetCustomFillColor and Shape.SetCustomLineColor.

Example

> Wherever type Color is referenced in the following example, the type is SoftArtisans.OfficeWriter.ExcelWriter.Color. If you are using ExcelWriter in a class that imports the .NET framework class System.Drawing (added by default by when a new webform is added to a VS.NET project), you will get an "ambiguous reference" error at compile-time because System.Drawing also contains a type Color. The solution is to either remove the System.Drawing import statement, or use the fully qualified name of ExcelWriter's Color object.

```
private void SetupColors(Workbook wb)
{

     //--- Workbook.Palette returns the Palette object
     Palette pal = wb.Palette;

     //--- GetClosestColor finds the color in the palette
     //--- that most closely matches the specified Red,
     //--- Green, and Blue values.
     Color clrDarkBlue = pal.GetClosestColor(0, 0, 255);
     Color clrTitleCell = pal.GetClosestColor(100, 100, 255);

     try
     {
          //--- GetColor will find the exact color you request.
          //--- If the color does not exist in the palette
          //--- it will throw IllegalArgumentException.
          Color clrDarkGreen = pal.GetColor(51, 153, 102);
     }
     catch
     {
          //--- The color you requested was not found in the palette
     }

     //--- GetColorAt gets the color from the palette at
     //--- the specified index.  This retrieves the second
     //--- color from the palette.
     Color clrComment = pal.GetColorAt(1);

     //--- SetColor and SetColorAt are for changing
     //--- the colors in the palette. Specify a Color to
     //--- change for SetColor, the index of the color for
     //--- SetColorAt, and then the RGB values for the desired
     //--- appearance.
     pal.SetColor(clrDarkBlue, 0, 0, 200);
     pal.SetColorAt(0, 0, 255, 0);

     //--- The Color object has built-in colors
     Color clrSysGreen = Color.SystemColor.Green;
     Color clrSysBlack = Color.SystemColor.Black;
}

private void StylizeHeaderCharacters(Workbook wb, Worksheet ws)
{
     Palette pal = wb.Palette;

     //--- Get a reference to cell A1
     Cell cellHeader = ws[0, 0];

     Color clrDarkBlue = pal.GetClosestColor(0, 0, 255);
     Color clrDarkGreen = pal.GetColor(51, 153, 102);

     //--- Stylize the first word, starting from the
     //--- first character, ending at the 10th
     cellHeader.GetCharacters(0, 10).Font.Size = 12;

     //--- Format the font for the next two words in the cell
     cellHeader.GetCharacters(11, 12).Font.Color = clrDarkBlue;
     cellHeader.GetCharacters(24, 12).Font.Color = clrDarkGreen;
}
```

Areas and Ranges

An **Area** object represents a rectangular set of cells. To create an Area object, call one of the following methods:

- `[Worksheet.CreateArea(Int32, Int32, Int32, Int32)]`

This method takes the 0-based indexes of the area's first row and column, and the number of rows and columns to include in the area.

- `[Worksheet.CreateArea(String)]`

This method takes a formula representing the area, for example, "=A1:G10". The formula is relative to the current worksheet.

- `[Worksheet.CreateAreaOfColumns(Int32, Int32)]`

This method creates a range containing one rectangular area.

> This method has been deprecated in favor of using Worksheet.GetColumnProperties(Int32) to get and edit column properties.

- `[Worksheet.CreateAreaOfRows(Int32, Int32)]`

This method creates a range containing one rectangular area.

> This method has been deprecated in favor of using Worksheet.GetRowProperties(Int32) to get and edit row properties.

A **Range** is a collection of areas. The areas in a range may be non-adjacent, and a range can include areas in different worksheets. To create a Range (without a name), call one of the following methods:

- `[Workbook.CreateRange(String)]`

This method takes a formula representing the range, for example "=Sheet1!A1:G10" defines a range containing one area and "=Sheet1!B$12:$H$21 Sheet1!$F$18:$K$29 Sheet1!D$16:$M$21" defines a range containing three areas. The formula must be three-dimensional (i.e., it must specify the sheet or sheets).

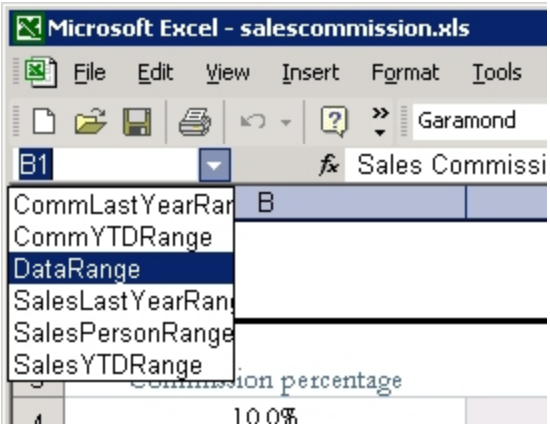
- `[Workbook.CreateRange(Area())]`

This method takes an array of Area objects representing the range.

- `[Worksheet.CreateRange(String)]`

This method takes a formula representing the range, for example "=Sheet1!A1:G10" defines a range containing one area and "=Sheet1!B$12:$H$21 Sheet1!$F$18:$K$29 Sheet1!D$16:$M$21" defines a range containing three areas. The formula must be three-dimensional (i.e., it must specify the sheet or sheets).

A **NamedRange** is stored in ExcelWriter's NamedRanges collections (Workbook.NamedRanges and Worksheet.NamedRanges) and is accessible after the workbook is saved. In Excel, named ranges are listed in the name box above the top-left corner of the worksheet.



To create a named range, call one of the following methods:

- `[Workbook.CreateNamedRange(String, String)]`

  This method takes a formula representing the range, for example "=Sheet1!A1:G10", and a name for the range. The formula must be three-dimensional (i.e., it must specify the sheet or sheets).

- `[Workbook.CreateNamedRange(Area(), String)]`

  This method takes an array of Area objects representing the range and a name for the range.

- `[Worksheet.CreateNamedRange(Int32, Int32, Int32, Int32, String)]`

  This method returns a named range that contains one rectangular area.

- `[Worksheet.CreateNamedRange(String, String)]`

  This method takes a formula representing the range, for example "=Sheet1!A1:G10", and a name for the range. The formula must be three-dimensional (i.e., it must specify the sheet or sheets).

Importing Data to an Area

You can use an area as a set of target cells for imported data. The Area object's ImportData method allows you to import a block of data from a rectangular array or from an ADO.NET DataTable, DataView, or DataReader. To import data to an area:

1. Create an area in a worksheet.

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheets[0];
Area targetArea = ws.CreateArea(4, 4, 15, 6);
```

2. Get a rectangular array, DataView, DataReader, or - as in the following example - DataTable to use as the data source.// ADO.NET code to get a DataTable from a query

```
DataTable employeeDt = new DataTable();
using(SqlConnection conn = new SqlConnection(connString))
{
 string employeeSQL = "SELECT FirstName + ' ' + LastName As Name " +
     "FROM Employee";
 SqlCommand cmdEmployee = new SqlCommand(employeeSQL, conn);
 SqlDataAdapter adptEmployee = new SqlDataAdapter(cmdEmployee);
 adptEmployee.Fill(employeeDt);
}
```

3. Call ImportData to import the data. The method returns a new area containing the imported values.

```
importedValues = targetArea.ImportData(employeeDt);
```

For more information, see Importing Data.

Applying Formatting to Areas and Ranges

Defined styles can be assigned to cells, rows, columns, areas, and ranges. To assign a style to a range or area:

1. Create a style.

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheets[0];
Style styleTotalRow = wb.CreateStyle();
styleTotalRow.NumberFormat = "$#.##0";
styleTotalRow.Font.Italic = true;
styleTotalRow.Font.Bold = true;
```

2. Define an area or range, for example:

```
Area areaTotalRow = ws.CreateArea(24, 0, 1, 3);
```

3. Set or apply the style.

```
areaTotalRow.Style = styleTotalRow;
```

For more information on setting and applying styles, see Styles.

Modifying an Existing Spreadsheet

ExcelApplication can open and modify an existing spreadsheet. The new spreadsheet can either be saved to disk (overwriting the original file or saved with a new name) or streamed to the browser. This allows you to use a preset format for similar spreadsheets, rather than recreate the format for each. Alternatively, use an ExcelWriter template.

All features in the original workbook - including macros, charts, PivotTables, and VBA - will be preserved in the new workbook.

How to Modify an Existing Spreadsheet

To modify an existing spreadsheet:

1. Create an instance of ExcelApplication:

```
ExcelApplication xla = new ExcelApplication();
```

2. Open an Excel workbook from a file path or a System.IO.Stream:

```
//--- Open a workbook from a file path:
Workbook wb = xla.Open(@"C:\Reports\Report.xls");
```

Or:

```
//--- Open a workbook from a Stream:
FileStream fs = new FileStream(@"C:\Sales\2003\June.xls", FileMode.Open);
Workbook wb = xla.Open(fs);
```

3. Return a Worksheet object:

```
Worksheet ws = wb.Worksheet[0];
```

4. Modify the worksheet. For example, add a cell value:

```
ws.Cells[0, 0].Value = "Welcome to SoftArtisans OfficeWriter";
```

5. Save the modified file with a new name, or stream it to the client:

```
//--- Save the file on the server:
xla.Save(wb, @"C:\temp\out.xls");
```

Or:

```
//--- Stream the file to the client via HTTP:
//--- false means Open In Excel, true means Open In Browser (IE only)
xla.Save(wb, Page.Response, "out.xls", false);
```

For more information on save options, see Output Options.

Importing Data

ExcelApplication'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 Worksheet and Area. The method returns a Area object representing a the set of cells that contain the imported data.

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

> ⚠ ImportData does not insert values into the worksheet and push existing values down. The imported data will overwrite existing values in the target cells.

Importing from a Database

You can import values from a database to a worksheet 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 worksheet:

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 Employee";

     DataTable dt = new DataTable();

     //--- "connString" is a SQL connection string
     using(SqlConnection conn = new SqlConnection(connString))
          new SqlDataAdapter(EmployeeSQL, conn).Fill(dt);

     return dt;
}
```

2. Pass the ADO.NET DataTable to ImportData:

```
//--- Import to a worksheet and pass ImportData
//--- the cell at which to start entering the data.
Cell CellB5 = Sheet1.Cells[4, 1];
Area importedValues = Sheet1.ImportData(dt, CellB5);

//--- Or: Import to a defined area.
Area targetArea = Sheet1.CreateArea(4, 4, 15, 6);
Area importedValues = targetArea.ImportData(dt);
```

Importing from an Array

To import values from a two-dimensional array to your worksheet:

1. Create a rectangular array, for example:

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

2. Pass the array to ImportData:

```
//--- Import to a worksheet and pass ImportData
//--- the cell at which to start entering the data.
Cell CellB5 = Sheet1.Cells[4, 1];
Area importedValues = Sheet1.ImportData(arrayData, CellB5);

//--- Or: Import to a defined area.
Area targetArea = Sheet1.CreateArea(4, 4, 15, 6);
Area importedValues = targetArea.ImportData(arrayData);
```

Customizing Your Data Import

The DataImportProperties class contains a set of properties that are used when importing data to cells in a worksheet. 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 multipleImportData calls.

To customize a data import using a DataImportProperties object:

1. Create a DataImportProperties object with the Workbook.CreateDataImportProperties() method:

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
DataImportProperties importProps = wb.CreateDataImportProperties();
```

2. Set one or more data import propeties:

```
//--- Truncate imported data rows and columns that do
//--- not fit within the target rows and columns in the spreadsheet.
importProps.Truncate = true;

//--- By default, when data is imported from a two-dimensional
//--- array, the data will be entered as [row][column]. If
//--- Transpose is set to true, the data will be entered as
//--- [column][row].
importProps.Transpose = true;
```

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

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

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

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheets[0];
DataImportProperties importProps = wb.CreateDataImportProperties();

//--- UseColumnNames will import column names in addition to data
importProps.UseColumnNames = true;

//--- Import to a worksheet and pass ImportData
//--- the cell at which to start entering the data.
Cell CellB5 = ws.Cells[4, 1];
String[] colNames = {"Name", "LastName", "Title"};
Area importedValues = ws.ImportData(arrayData, colNames,
     CellB5, importProps);

//--- Or: Import to a defined area.
Area targetArea = ws.CreateArea(4, 4, 15, 6);
String[] colNames = {"Name", "LastName", "Title"};
Area importedValues = targetArea.ImportData(arrayData, colNames,
     importProps);
```

Charts in ExcelApplication

ExcelApplication supports all Excel chart types, and allows you to insert a chart in a worksheet, create a chart sheet (a worksheet that only contains a chart).

If you open an existing Excel file with ExcelApplication.Open, you can use ExcelWriter's charting objects to modify charts in the spreadsheet.

To create a simple column chart:

1. Get a Charts collection:

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheets[0];
Charts chrts = ws.Charts;
```

   The Charts collection contains all charts in a specified worksheet.

2. Create an anchor in the worksheet:

```
Anchor anch = ws.CreateAnchor(7, 4, 0, 50);
```

An anchor represents the position of a floating object (e.g., a chart) within a worksheet. The chart's top left corner will be placed at the anchor.

3. Create a blank chart of a specified type, at the anchor you created:

```
Chart chrt = chrts.CreateChart(ChartType.Column.Clustered, anch);
```

The ChartType class contains all available chart types (e.g., Column) and sub-types (e.g., Clustered).

4. Return a SeriesCollection object representing the set of data series in the chart:

```
SeriesCollection sc = chrt.SeriesCollection;
```

5. Set the range of category (x) axis values:

```
sc.CategoryData = "Sheet1!A3:C3";
```

6. Add a data series to the chart:

```
Series s = sc.CreateSeries("Sheet1!A25:C25");
```

The formula passed to CreateSeries represents cells that contain the source values for the new data series.

Example: Creating a Chart

```
using System;
using SoftArtisans.OfficeWriter.ExcelWriter;

class ChartDemo : System.Web.UI.Page
{
     protected void Page_Load(object sender, System.EventArgs e)
     {
          //--- Create ExcelApplication, a Workbook, and a Worksheet
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];

          //--- Put some values into the cells that the
          //--- chart will reference
          System.Random rand = new System.Random();

          //--- Headers
          ws.Cells[0, 0].Value = "Month";
          ws.Cells[0, 1].Value = "Sales";

          //--- Month and sales
          ws.Cells[1, 0].Value = "Jan";
          ws.Cells[1, 1].Value = rand.Next(1000);
          ws.Cells[2, 0].Value = "Feb";
          ws.Cells[2, 1].Value = rand.Next(1000);
          ws.Cells[3, 0].Value = "Mar";
          ws.Cells[3, 1].Value = rand.Next(1000);

          //--- Create an Anchor on Cell B6
          Anchor anch = ws.CreateAnchor(4, 2, 0, 0);

          //--- Create the chart
          Chart chrt = ws.Charts.CreateChart(ChartType.Column.Clustered, anch);

          //--- Set series collection
          string seriesFormula =
               String.Format("={0}!{1}:{2}", ws.Name,
               ws.Cells[1, 1].Name, ws.Cells[3, 1].Name);
          Series srs = chrt.SeriesCollection.CreateSeries(seriesFormula);
          srs.Name = "Sales";

          //--- Set category data
          chrt.SeriesCollection.CategoryData =
               String.Format("{0}!{1}:{2}", ws.Name,
               ws.Cells[1, 0].Name, ws.Cells[3, 0].Name);

          xla.Save(wb, Page.Response, "Charts.xls", false);
     }
}
```

Modifying an Existing Chart

Code sample: Modifying an Existing Chart

[C#] | [VB.NET]
When you use ExcelApplication to open a spreadsheet, you can access charts through their names or titles. Chart.getTitle returns a ChartText object representing the title region. ChartText.getText returns the title text. The following function takes a chart's title and returns a Chart object representing the chart. This code can be useful if you want to locate a specific chart in a worksheet but only know its title text:

```
private Chart FindChart(string title)
{
     for (int iChart = 0; iChart < ws.Charts.Count; iChart++)
     {
          Chart chrt = ws.Charts[iChart];
          if (chrt.Title.Text == title)
               return chrt;
     }
     return null;
}
```

The following example uses the FindChart code to locate a specific chart in a worksheet. The chart series collection is then cleared and re-set to the data imported from a database:

```
private void PopulateMainChart()
{
     //--- Get the data from the database.
     DataTable[] dtArr = GetCategoryQuarterlySales();

     //--- Find the main chart based on its title text, and get
     //--- a reference to it.
     Chart mainChart = FindChart("Quarterly Sales 2003");

     //--- Clear all the existing series objects from the collection
     SeriesCollection seriesCol = mainChart.SeriesCollection;
     while (seriesCol.Count > 0)
          seriesCol.Remove(0);

     //--- Each DataTable has a single row of data.
     int iRow = 29;
     for (int i = 0; i < dtArr.Length; i++)
     {
          DataTable dt = dtArr[i];

          //--- Import data from the DataTable.
          ws.ImportData(dt, ws[iRow, 1]);

          //--- Add the imported data as a new Series object
          //--- in the chart's collection.  There will be a variable
          //--- number of series objects depending on which categories
          //--- were selected for display.
          Area a = ws.CreateArea(iRow, 2, 1, 4);
          Series srs = seriesCol.CreateSeries(a);
          srs.NameFormula = ws[iRow, 1].Name;

          iRow++;
     }

     //--- Re-set the category data.  The size will vary
     //--- depending on the selected categories.
     seriesCol.CategoryData =
          ws.CreateArea(29, 1, dtArr.Length, 1).Dimensions;

     //--- Add a legend to the chart if desired
     if (bLegend)
     {
          mainChart.Legend.Visible = true;
          mainChart.Legend.Location = Legend.LegendLocation.Top;
     }
     else
     {
          //--- The legend is hidden in the template workbook.
          //--- Hide it again in case it's made visible.
          mainChart.Legend.Visible = false;
     }
}
```

Drawing Objects

In addition to charts, ExcelApplication allows you to add the following drawing objects to a worksheet:

- **Comments**
  A comment is a note attached to a cell, or group of merged cells, that is separate from other cell content. A comment is represented by a Comment object.
- **Pictures**
  A picture is a JPEG, PNG, BMP, or GIF image that is inserted in the worksheet. A picture is represented by a Picture object.
- **Shapes**
  A shape may be one of Excel's ready-made AutoShapes or the display area of a comment or picture. A shape is represented by a Shape object.

Adding a Comment to a Worksheet

A comment must be associated with a cell. Once created, comments may not be moved between cells. To create a comment:

1. Use the property Cell.Comment to return a Comment object.

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheets[0];

Cell cellB5 = ws.Cells["B5"];
Comment comm = cellB5.Comment;
```

2. Add text to the comment.

```
comm.Text = "Figures for March are projected.";
```

3. Specify the author of the comment.

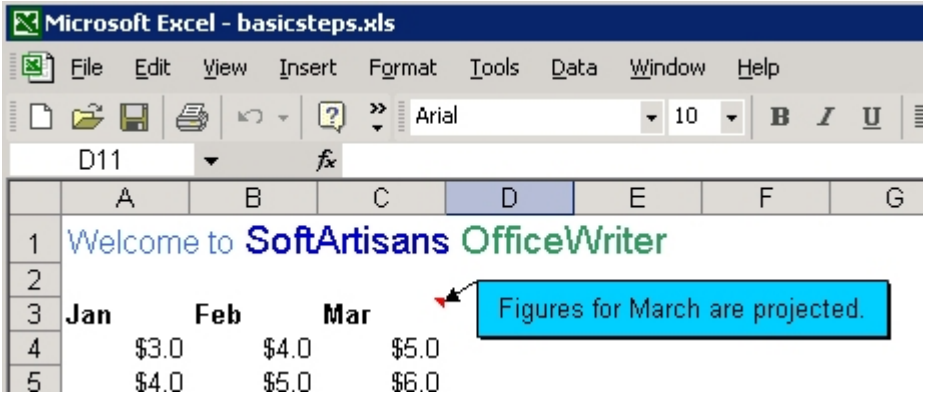
```
comm.Author = "John Doe";
```

4. Set whether the comment will be visible when the document is opened in Excel. By default, the comment will be hidden, and will be displayed only if the user hovers over the comment's cell.

```
comm.Visible = true;
```

5. To modify the comment's display area, return its Shape object and set Shape properties, for example:

```
Shape commShape = comm.Shape;
commShape.FillColor = oColor;
commShape.FitToText = true;
```



A comment is a note attached to a cell - or group of merged cells - that is separate from other cell content.

Creating a Picture

Pictures in a worksheet are stored in the Pictures collection. To add a picture to a worksheet:

1. Return a Pictures collection.

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheets[0];
Pictures pics = ws.Pictures;
```

2. Create a Picture object from a JPEG, PNG, BMP, or GIF image and insert it at a specified anchor in the worksheet. The anchor represents the position of the top-left corner of the picture in the worksheet.

```
string imgPath = @"c:\images\owlogo.jpg";
Anchor anch = ws.CreateAnchor(0, 7, 0, 0);
Picture pic = pics.CreatePicture(imgPath, anch);
```

3. To modify the size, position, or formatting of a Picture, return its Shape object and set Shape properties, for example:

```
Shape picShape = pic.Shape;
picShape.Rotation = 45;
```

Creating an AutoShape

Excel's ready-made AutoShapes are represented by the Shapeclass. To create an AutoShape:

1. Return a Shapes collection.

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheets[0];
Shapes shps = ws.Shapes;
```

2. Create a shape of a specified type and insert it at an anchor in the worksheet.

```
Anchor anch = ws.CreateAnchor(0, 7, 0, 0);
Shape shpHeart = shps.CreateShape(ShapeType.Heart, anch);
```

The ShapeType class contains the Excel AutoShapes supported by ExcelWriter. The specified anchor represents the position of the top-left corner of the shape in the worksheet.

Output Options for ExcelApplication

Using the ExcelApplication object's Save methods, you can:

- Save the generated Excel workbook to disk,
- Stream the Excel workbook to the client,
- Write the Excel spreadsheet to a specified output stream, or,
- Return the workbook as an ExcelTemplate object.

When ExcelWriter 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 one of the following methods:

```
void Save(Workbook workbook, string fileName)
void Save(Workbook workbook, string fileName)
```

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

Example

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
xla.Save(wb, @"C:\Sales2003\June.xls");
```

Stream to Client

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

```
void Save(Workbook workbook, System.Web.HttpResponse response, string fileName)
void Save(Workbook workbook, System.IO.Stream stream)
void Save(Workbook workbook, System.Web.HttpResponse response, string fileName, bool openInBrowser)
void Save(Workbook workbook, System.Web.HttpResponse response, string fileName, bool openInBrowser,
string contentType)
```

Browsers other than Internet Explorer cannot embed an Excel file in the browser window. When the generated spreadsheet 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 Excel or another spreadsheet application.

Internet Explorer can display an Excel file in the browser window. When the generated spreadsheet 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 spreadsheet application. If openInBrowser is set to true, the response content-disposition header is set to open the file in the browser window.

Example

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
xla.Save(wb, Page.Response, "workbook.xls", false);
```

Write to Stream

To write the generated spreadsheet to a System.IO.Stream, call the following method:

```
void Save(Workbook workbook, System.IO.Stream stream)
```

Example

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
FileStream fstream = new FileStream(@"C:\temp\outfile.xls",
FileMode.Create);
xla.Save(wb, fstream);
```

Return a Template

You can use ExcelApplication to create a spreadsheet with data markers (i.e., a template) and pass the spreadsheet to ExcelTemplate to populate the data markers. In this case, do not call Save. Instead, pass your Workbook object to ExcelTemplate's Open method:

```
void ExcelTemplate.Open(ExcelApplication excelApplication, Workbook workbook)
```

Example

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
ExcelTemplate xlt = new ExcelTemplate();
xlt.Open(xla, wb);
```

Passing ExcelApplication to ExcelTemplate

Passing ExcelApplication to ExcelTemplate

ExcelWriter allows you to generate a spreadsheet from script alone - using the ExcelApplication object - or from a template spreadsheet and a script, using ExcelTemplate. You can also use ExcelApplication and ExcelTemplate together. This section shows you how to generate a template spreadsheet with ExcelApplication and pass the spreadsheet to ExcelTemplate which then populates the template's data markers.

1. Use ExcelApplication to create a worksheet and enter data marker strings in the worksheet's cells:

   ```
   ExcelApplication xla = new ExcelApplication();
   Workbook wb = xla.Create();
   Worksheet ws = wb.Worksheets[0];

   for (int i = 0; i < colCount; i++)
   {
        //--- Form the current data marker.
        //--- Data markers are in this format:
        //--- %%=[dsrcName].[colName]
        string curMarker = String.Format("%%={0}.{1}", dsrcName,
             selectedCols[i].ToString());
        ws[firstRow, firstCol + i].Value = curMarker;
   }
   ```

   A data marker is a cell value beginning with %%= or %%=$ that specifies a variable, an array, or a database column, to insert in the spreadsheet column containing the marker. The data marker may be followed by a modifier. For example, to include column headers in the populated data marker column, the data marker should include the (fieldname) modifier:

   ```
   %%=DataSource.EmployeeID(fieldname)
   ```

   For more information, see Creating Data Markers.

2. To open the workbook you created as an ExcelTemplate, pass your ExcelApplication and Workbook objects to ExcelTemplate's Open method:

```
ExcelTemplate xlt = new ExcelTemplate();
xlt.Open(xla, wb);
```

3. Connect to a database and execute a query to return a DataTable, DataView, or DataReader to use a data source for template data markers, for example:

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

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

     return dt;
}
```

Alternatively, the data source for the template may be a rectangular array.

4. Set the returned DataTable, DataView, or DataReader as the data source for template data markers, and call Process to populate the data markers with data source values:

```
xlt.SetDataSource(dt, dsrcName);
xlt.Process();
```

5. Call Save to save the template on the server, return it in memory, or - as in the following example - stream it to the browser:

```
xlt.Save(Page.Response, "Report.xls", false);
```

Example

```
using SoftArtisans.OfficeWriter.ExcelWriter;

class AppToTemplate : System.Web.UI.Page
{
     protected void Page_Load(object sender, System.EventArgs e)
     {
          //--- Create ExcelApplication, a Workbook, and a Worksheet
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];

          //--- Write ExcelTemplate data markers into the workbook
          ws.Cells[0, 0].Value = "%%=DSrcName.#1(fieldname)";
          ws.Cells[0, 0].Style.Font.Bold = true;
          ws.Cells[1, 0].Value = "%%=DSrcName.#1";

          ws.Cells[0, 1].Value = "%%=DSrcName.#2(fieldname)";
          ws.Cells[0, 1].Style.Font.Bold = true;
          ws.Cells[1, 1].Value = "%%=DSrcName.#2";

          //--- Open the Workbook as ExcelTemplate
          ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(xla, wb);

          //--- Bind data using ExcelTemplate SetDataSource
          string[,] values = {{"New York", "NY"},
               {"Miami", "FL"},
               {"Boston", "MA"}};
          string[] names = {"City", "State"};
          xlt.SetDataSource(values, names, "DSrcName");
          xlt.Process();
          xlt.Save(Page.Response, "AppToTemplate.xls", false);
     }
}
```

Code Sample: Passing ExcelApplication to ExcelTemplate

[C#] | [VB.NET]

Formatting Headers and Footers

To create and format spreadsheet headers and footers, use the HeaderFooterSection object, which can be retrieved using the PageSetup.GetHeader(HeaderFooterSection.Section) and PageSetup.GetFooter(HeaderFooterSection.Section) methods.

> To see spreadsheet headers and footers in Excel, open the **File** menu, select **Page Setup**, and select the **Header/Footer** tab.

To format a header or footer in script, include formatting codes within the SetContent() call, for example:

```
using SoftArtisans.OfficeWriter.ExcelWriter;

public class HeaderFooterDemo : System.Web.UI.Page
{
     protected void Page_Load(object sender, System.EventArgs e)
     {
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet sheet1 = wb.Worksheets[0];
          PageSetup pgsetup = sheet1.PageSetup;
          HeaderFooterSection leftHeader =
               PageSetup.GetHeader(HeaderFooterSection.Section.Left);
          leftHeader.SetContent("Document name: &F");
          HeaderFooterSection centerHeader =
               PageSetup.GetHeader(HeaderFooterSection.Section.Center);
          centerHeader.SetContent("Basic Report");
          HeaderFooterSection rightHeader =
               PageSetup.GetHeader(HeaderFooterSection.Section.Right);
          rightHeader.SetContent("Generated by OfficeWriter");
          HeaderFooterSection leftFooter =
               PageSetup.GetFooter(HeaderFooterSection.Section.Left);
          leftFooter.SetContent("Created on date &D");
          HeaderFooterSection centerFooter =
               PageSetup.GetFooter(HeaderFooterSection.Section.Center);
          centerFooter.SetContent("Page &P of &N");
          HeaderFooterSection rightFooter =
               PageSetup.GetFooter(HeaderFooterSection.Section.Right);
          rightFooter.SetContent("Create at time &T");
     }
}
```

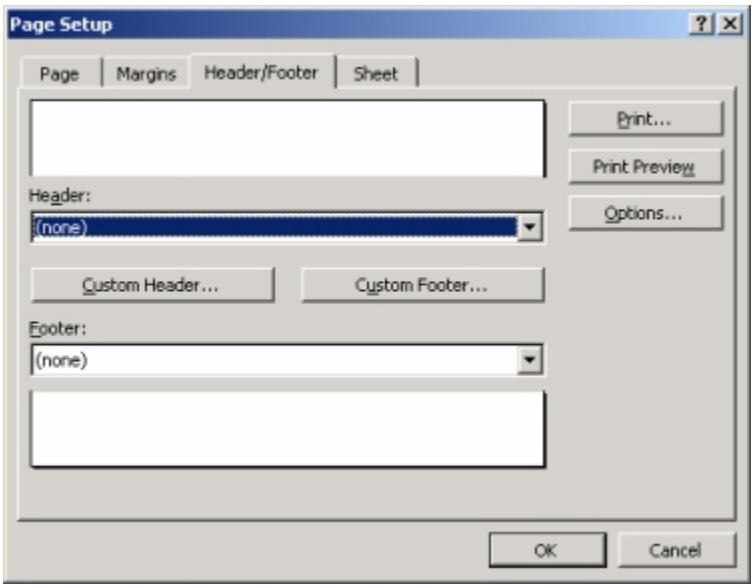
Formatting Codes for Headers and Footers	
&E	Turns double-underline on or off.
&X	Turns superscript on or off.
&Y	Turns subscript on or off.
&B	Turns bold on or off.
&I	Turns italic on or off.
&U	Turns underline on or off.
&S	Turns strikethrough on or off.
&D	Prints the current date.
&T	Prints the current time.
&F	Prints the name of the document.
&A	Prints the name of the worksheet.
&P	Prints the page number.
&P+*number*	Prints the page number plus the specified number.
&P-*number*	Prints the page number minus the specified number.
&&	Prints a single ampersand.
&"*font name*"	Prints the characters that follow in the specified font.
&*nn*	Prints the characters that follow in the specified font size. Use a two digit number to specify a size in points.
&N	Prints the total number of pages in the document.

Formatting Headers and Footers in Microsoft Excel

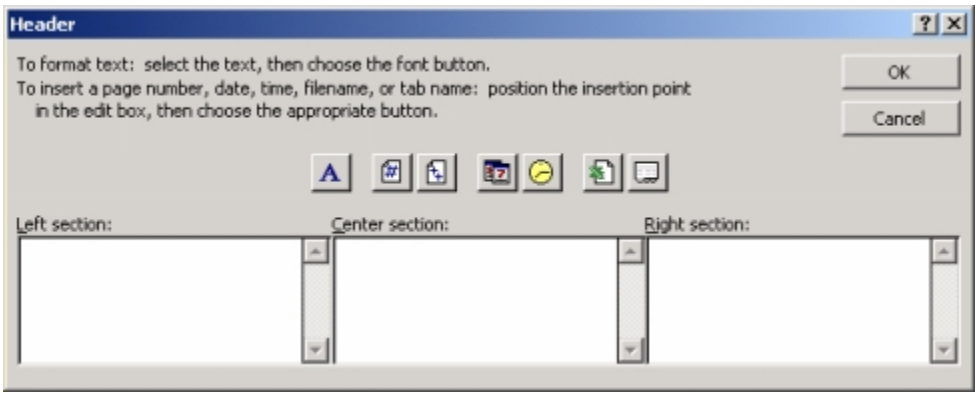
When opening an existing Excel spreadsheet, ExcelWriter will preserve the header and footer formatting created in Excel. However, if you change the header/footer text in your ExcelWriter script and do not include Formatting Codes, the formatting created in Excel will be overwritten and default formatting will be applied. To preserve header/footer formatting created in Excel, do not change header/footer text in script.

To format spreadsheet headers and footers in Excel and preserve the formatting in ExcelWriter:

1. In Microsoft Excel, open the **File** menu and select **Page Setup...**
2. Select the **Header/Footer tab**.
3. Select a header and/or footer from the drop-down list.



Or, create a custom header/footer.



4. Save the spreadsheet.
5. In an ExcelWriter script, open the spreadsheet containing headers/footers. Use either ExcelTemplate.Open or ExcelApplication.Open to open the spreadsheet.
6. Modify the spreadsheet in script. **Do not change the text of headers and footers**; if you do, the original formatting will be overwritten and default formatting will be applied.

Protecting Your Worksheet

There may be times when you want to lock your Excel worksheet so that end users cannot edit or manipulate the data in a worksheet. You can implement worksheet protection using ExcelWriter by calling the method Worksheet.Protect().

Worksheet protection is meant to lock data for the purpose of presentation only. Worksheet protection should not be confused with **encryption**, and it should not be used to hide or protect sensitive data. When worksheet protection is activated, users can see the worksheet and all the data within it, but will be prevented from altering the content or formatting the cells.

Encryption can be performed on the workbook level, as shown below:

Workbook Encryption

```
//--- To encrypt a workbook, use the Workbook.EncryptPassword property:
ExcelApplication xlw = new ExcelApplication();
Workbook wb = xlw.Create();
wb.EncryptPassword = "myPassword";
xlw.Save(wb, outPath);
```

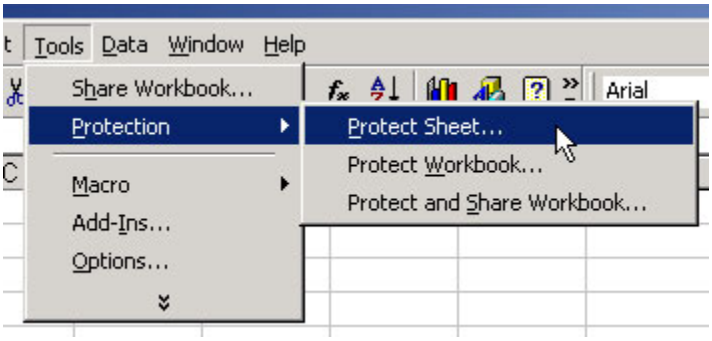
How to Activate Worksheet Protection with ExcelWriter

Using ExcelWriter, there are two ways to write-protect worksheets in your workbook:

1. Protect in Microsoft Excel and Open in ExcelWriter
2. Protect in ExcelWriter Code

Protect in Microsoft Excel and Open in ExcelWriter

Create a workbook in Microsoft Excel, open the **Tools** menu, and select **Protection -> Protect sheet**. Then, open the spreadsheet with ExcelWriter script, using either ExcelTemplate or ExcelApplication.Open. ExcelWriter will preserve all existing spreadsheet settings, including worksheet protection.



Protect in ExcelWriter Code

Use the method Worksheet.Protect to active worksheet protection. Pass the method a password for unlocking the worksheet in Excel. Pass "" or nullto protect the worksheet without a password.

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheet[0];
ws.Protect("MyPassword");
```

Removing Worksheet Protection

To remove worksheet protection in Microsoft Excel,

1. From the **Tools** menu, select **Protection -> Unprotect Sheet**.
2. If a password was set, enter it and click **Ok**.

To remove worksheet protection with ExcelWriter, call Worksheet.Unprotect:

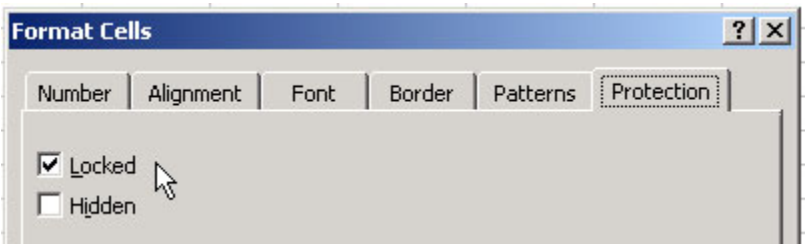
```
ws.Unprotect();
```

Protecting Specific Cells

When you activate worksheet protection, all of the cells in the worksheet will be locked by default. However, you can unlock cells within a protected worksheet, allowing the user to fill in or modify some fields.

To unlock cells in Microsoft Excel:

1. Select a cell or cells to unlock.
2. From the **Format** menu select **Cells...**
3. Select the **Protection** tab.
4. Uncheck **Locked**.



With ExcelWriter, you can unlock cells by setting Style.CellLocked to false and applying the style to an individual cell. The following example protects all cells except for A1 and B1:

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Worksheet ws = wb.Worksheet[0];

//--- Turn on worksheet protection.
ws.Protect("MyPassword");

//--- Create an "unlocked" style.
Style unlockedStyle = wb.CreateStyle();
unlockedStyle.CellLocked = false;

//--- Apply the style.
ws.Cells["A1"].Style = unlockedStyle;
ws.Cells["B1"].Style = unlockedStyle;
```

Copying Data From a Worksheet

ExcelApplication's CopyPaste method allows you to copy data from an external worksheet. It can also be used to copy data from a worksheet within the same workbook. CopyPaste is a method of the Worksheet object and should be called from the destination worksheet. CopyPaste uses a CopyPasteProperties object to store information on precisely how data should be copied. Options include: values, formulas, formatting, merged cells, comments, height and width.

Step 1: Setting up your code to copy from an external worksheet:

In the code below, a source worksheet is opened and a destination worksheet is created. The CopyPaste method takes an Area object to represent the area to be copied, so this is created first.

```
private ExcelApplication xlw;
private Workbook wb, wb2;
private Worksheet wsSrc, wsDest;

private void GenerateWorkbook()
{
 //--- Store the path to the source workbook
 string strPathSrc = @"C:\\WorkDirectory\\MySourceWorkbook.xls";

 //--- Create an Excelwriter Application object
 xlw = new ExcelApplication();

 //--- Use the Application object to create a workbook
 //--- that will contain the destination worksheet
 wb = xlw.Create();

 //--- Specify the destination worksheet
 wsDest = wb.Worksheets[0];
 wsDest.Name = "DestinationSheet";

 //--- Open the workbook that contains the (source) worksheet to be copied
 wb2 = xlw.Open(strPathSrc);

 //--- Specify the worksheet that will be the source of the data
 wsSrc = wb2.Worksheets[0];

 //--- You must first create an area that represents the information
 //--- in the source worksheet that you want to copy.
 Area areaSrc = wsSrc.CreateArea("A1:C3");
```

Step 2: Creating a CopyPasteProperties object

Now that the data to be copied is mapped to an Area object, a CopyProperties object should be created from the destination workbook. The CopyProperties object is used to specify precisely what within the area should be copied. In the code below, only the values will be copied.

```
CopyPasteProperties properties =
     wb.CreateCopyPasteProperties(CopyPasteProperties.CopyPasteType.ValuesOnly);
```

The ValuesOnly type is a convenience type. The same result could be achieved by using the None type and adding a second line of code where the CopyValues property was set to true.

```
CopyPasteProperties properties =
     wb.CreateCopyPasteProperties(CopyPasteProperties.CopyPasteType.None);
 properties.CopyValues = true;
```

Using the None type allows you to set Copy properties individually. The available properties include:

```
[CopyPasteProperties].CopyColumnWidth = [true|false];

 [CopyPasteProperties].CopyComments = [true|false];

 [CopyPasteProperties].CopyFormatting = [true|false];

 [CopyPasteProperties].CopyFormulas = [true|false];

 [CopyPasteProperties].CopyMergedCells = [true|false];

 [CopyPasteProperties].CopyRowHeight = [true|false];

 [CopyPasteProperties].CopyValues = [true|false];
```

Besides None, other CopyPasteType values are available, saving you unnecessary lines of code:

CopyPasteType	Sets the following CopyPasteProperty values to true
None	No CopyPasteProperty values set.
All*	CopyComments, CopyFormatting, CopyFormulas, CopyMergedCells and CopyValues Note: CopyColumnWidth and CopyRowHeight are set to false to avoid unexpected results.
AllPlusRowHeightAndColumnWidth	CopyComments, CopyFormatting, CopyFormulas, CopyMergedCells, CopyValues, CopyColumnWidth and CopyRowHeight
FormulasAndFormatting	CopyFormulas and CopyFormatting
ValuesAndFormatting	CopyValues and CopyFormatting
ValuesFormulasAndFormatting	CopyValues, CopyFormulas and CopyFormatting
ValuesOnly	CopyValues

*Optionally, if you intend to use the CopyPasteType of All, you can save even more lines of code by using CopyPaste method without the properties object parameter. Omission of the properties object parameter will result in the assumption of the value of "All".

Step 3: Calling the CopyPaste method from the Destination Worksheet

To complete the code sample that is being built, only the CopyPaste method now needs to be called. You will pass it the address of the cell that represents the upper left corner of the location to paste and the Area and CopyPasteProperties objects that were created above.

```
//---Call the CopyPaste method from the destination worksheet.
     wsDest.CopyPaste("C3", areaSrc, properties);

     //---Open the spreadsheet in the browser to check the results
     xlw.Save(wb, Page.Response, "ExcelApp.xls", false);
}
```

Multilingual Support

Displaying Unicode Characters Where

Client and Server are Running the Same Language

ExcelWriter can be used to generate Excel spreadsheets which contain Unicode characters. When the client and server are running the same language, to correctly display Unicode characters in an Excel spreadsheet, set the charset property in the script that will display the spreadsheet to either:

- UTF-8, to correctly display characters in any language
  OR
- A specific language charset - like GB2312 for Simplified Chinese - to correctly display characters in a single language

For a complete list of charset values, see Character Set Recognition.

Displaying Non-Latin Alphabet Strings Where

Client and Server are not Running the Same Language

The ExcelWriter methods AnsiToUnicode and UnicodeToAnsi allow you to correctly display non-Latin alphabet strings, both when getting a value from a spreadsheet and displaying it in the browser, and when inserting a string from the client in a spreadsheet generated on the server. AnsiToUnicode and UnicodeToAnsi are methods of both ExcelApplication and ExcelTemplate.

AnsiToUnicode

If a client submits a non-Latin alphabet string to a server running a language different from the language of the client's HTML page, to display the string correctly in an Excel spreadsheet the string must be converted to Unicode. ExcelWriter's AnsiToUnicode method takes a non-Latin alphabet string and its language's code page and returns a Unicode string:

```
UnicodeString = AnsiToUnicode(AnsiString, CodePage)
```

To correctly display a non-Latin alphabet string in your ExcelWriter generated spreadsheet where the server's language and the string language are different:

1. Set the client-side HTML script's charset property to the string language's charset. For example, to set the charset to Hebrew, use:

```
<META HTTP-EQUIV="content-type" CONTENT="text/html;charset=windows-1255">
```

2. Use AnsiToUnicode to convert the string to Unicode. The following example gets a Hebrew string from an HTML form, converts the string to Unicode, and assign the string to cell E7:

```
Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
HebrewString = xlw.ansitounicode(Request.Form("FirstName"), 1255)
ws.cells("E7").value = HebrewString
%>
```

Note: If your ExcelWriter script will receive strings from the client in only one language, use that language's code page and charset. However, to correctly display values in any language, use the UTF-8 charset and 65001 code page.

For a complete list of charset values, see, Character Set Recognition.

UnicodeToAnsi

To get a non-Latin alphabet string from a spreadsheet, and display the string correctly in the browser from a server running a language different from the language of the client's HTML page, you must:

1. Set the ExcelWriter script's **charset** property to the string language's charset.
2. Use the UnicodeToAnsi method to convert the string to an Ansi encoded string that can be displayed correctly to the user.

UnicodeToAnsi takes a Unicode string and the string language's code page and returns an Ansi string:

```
AnsiString = UnicodeToAnsi(UnicodeString, CodePage)
```

In the following example, Cell E7 contains a Hebrew string represented in Unicode. The script converts the string from Unicode to Ansi, and displays the converted string in the browser. The Hebrew characters will be displayed correctly.

```
<META HTTP-EQUIV="content-type" CONTENT="text/html;charset=windows-1255">
<%
Set xlw = Server.CreateObject("Softartisans.ExcelWriter")...
response.write xlw.unicodetoansi(order.cells("E7").value, 1255)
```

Note: If your ExcelWriter script will get and display strings only in one language, use that language's code page and charset. However, to correctly display values in any language, use the UTF-8 charset and 65001 code page.

For a complete list of charset values, see, Character Set Recognition.

SharePoint Integration in Depth

Introduced in build 7.1.0.1855

- Exporting list as Excel spreadsheet
- SharePoint Solution - Excel Export Plus

Exporting list as Excel spreadsheet

Introduced in build 7.1.0.1855

- 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, ExcelWriter offers integration with SharePoint. This tutorial shows you how to provision a SharePoint custom list, then export the list items as an Excel spreadsheet. To a lesser extent, this tutorial also illustrates how to create a grouped spreadsheet using the ExcelTemplate object.

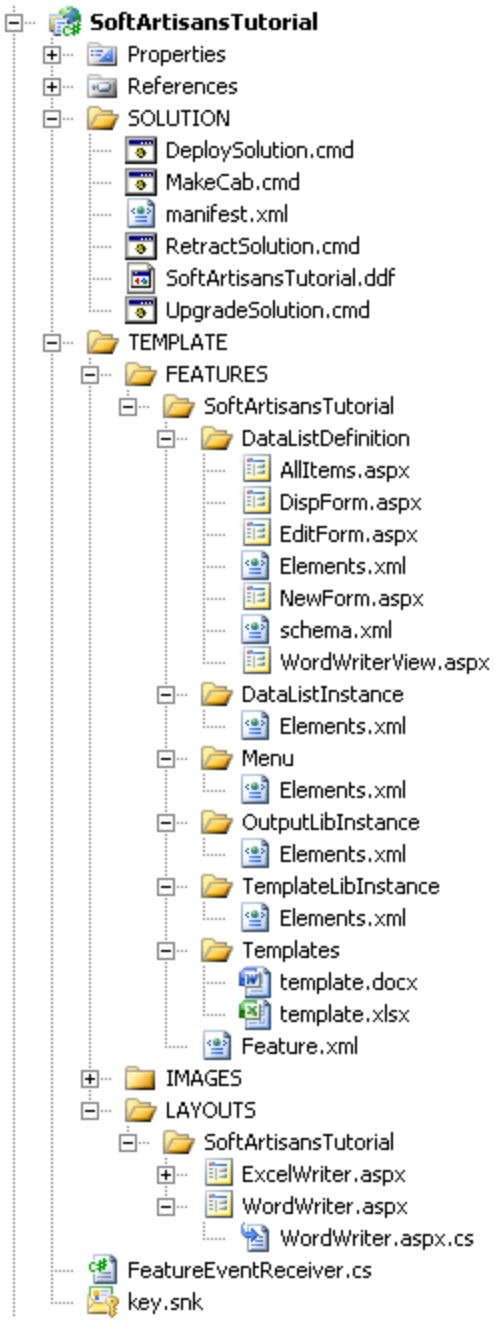
> The SharePoint integration and grouping features are only available in ExcelWriterEE. You must install ExcelWriter EE before running the tutorial.
>
> SharePoint integration makes use of extension methods, which require .NET 3.5.

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 ExcelWriter template,
- an empty document library to store the ExcelWriter-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 *ExcelWriterView*. 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 defines the fields shown in the view under the *<ViewFields>* node (line 29). We don't specify sort order in the view because the data are already pre-sorted.

```
<View BaseViewID="1" Type="HTML" WebPartZoneID="Main" DisplayName="ExcelWriterView" DefaultView="TRUE"
  <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>
    </OrderBy>
  </Query>
  <RowLimit Paged="TRUE">1000</RowLimit>
```

Step 3: Provision custom list instance

The Elements.xml file in the *DataListInstance* folder provisions an instance of the custom list, *SoftArtisans Data*, defined above. The *TemplateType* attribute of the list instance must match the type of the definition. 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 Excel 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 spreadsheet generated by ExcelWriter 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.MenuItemExcel"
  GroupId="ActionsMenu"
  Location="Microsoft.SharePoint.StandardMenu"
  RegistrationType="List"
  Sequence="1000"
  Title="Export to SoftArtisans OfficeWriter for Excel"
  ImageUrl="/_layouts/images/softartisanstutorial/actionicon_excel.gif">
  <UrlAction Url="~site/_layouts/SoftArtisansTutorial/ExcelWriter.aspx?ListId={ListId}"/>
</CustomAction>
```

Step 7: Create application page

The application page uses ExcelTemplate to export the items in the custom list as an Excel spreadsheet. 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 ExcelWriter to behave in a consistent and expected manner.

Instead of exporting the list of orders as a plain Excel file, as done by the out-of-the-box Excel export functionality, we want to generate a full-featured Excel report using ExcelWriter. The report should group the orders by year, by state within each year, by city within state, then finally by customer within city. Each grouping level should sub-total the order amount. We will accomplish this goal using the grouping functionality of ExcelTemplate. We will also add a simple bar chart showing the order total by year.

The steps for exporting the SharePoint list using ExcelWriter are as follows:

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

   ```
   ExcelTemplate xlt = new ExcelTemplate();
   ```

6. Load the template directly from the document library:

   ```
   xlt.Open(docLib, "template.xlsx");
   ```

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

7. Set the data source for the template:

   ```
   xlt.BindData(list.Views["ExcelWriterView"], list, "ds", xlt.CreateDataBindingProperties());
   ```

   The above BindData extension method takes an SPList as well as an SPView. The view allows us to refine the data returned from the list. Since we can bind multiple data sources to a template, we specify that the list data should populate data markers which reference the data source named "ds".

8. Call the ExcelTemplate.Process() method to allow ExcelWriter to populate the template:

```
xlt.Process();
```

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

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

xlt.Save(outputLib, "populated.xlsx", 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.

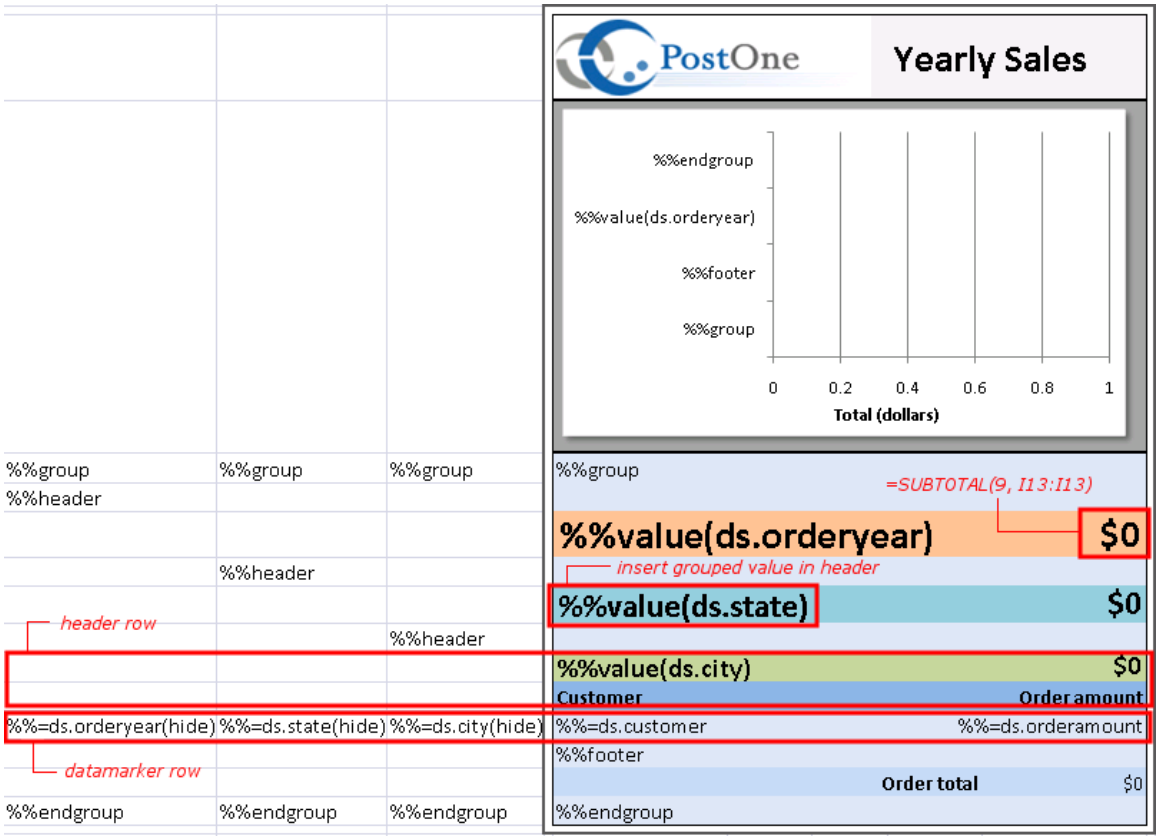
10. Instead of or in addition to saving the file to a document library, you can use other overloads of the ExcelTemplate.Save method. For example, to stream a copy of the spreadsheet to the browser:

```
xlt.Save(Response, "populated.xlsx", 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.

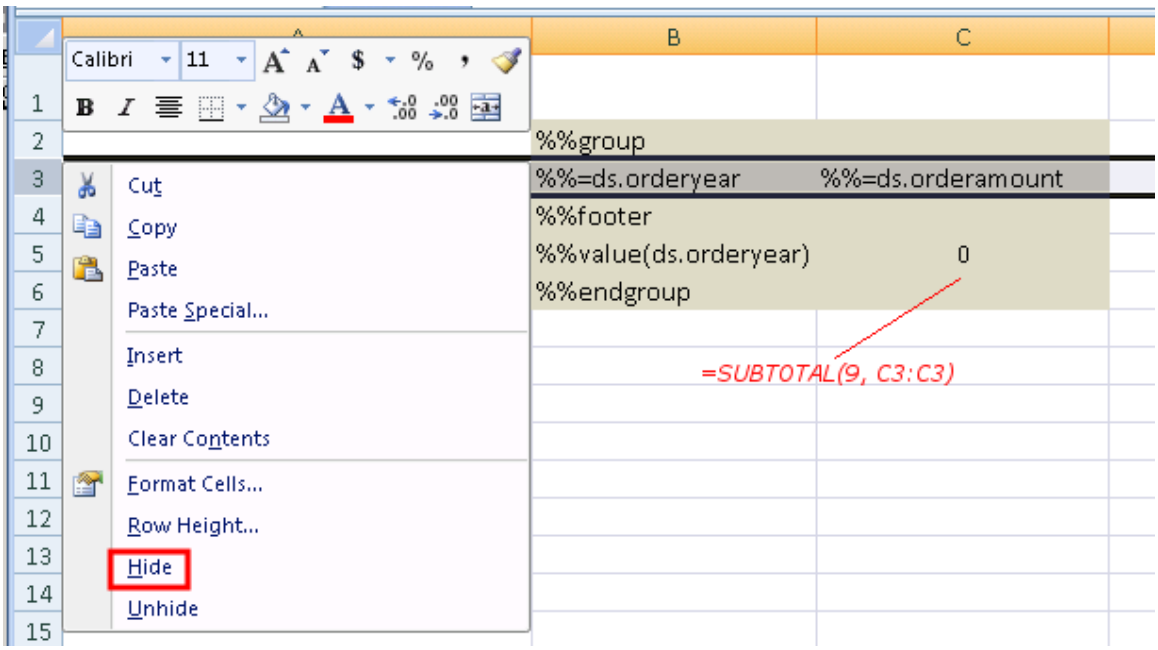


A grouped field is marked by the *%%group* and *%%endgroup* markers. Only one *%%endgroup%* marker is required, even if there are more than one grouped field. However, for clarity, we include an *%%endgroup* for each grouped field. In the template, we group over order year, then state,

then city, and finally customer. For aesthetic purposes, we want to display the grouped values for order year, state and city in a vertical orientation. To do so, we hide the corresponding data markers with the *(hide)* modifier. Then we show the hidden grouped values by adding a header row and using the *%%value(field)* marker.

To add an order amount sub-total for each group, we use Excel's *SUBTOTAL* formula. The formula should reference a range containing the data marker for the order amount, *%%=ds.orderamount*, which occupies cell *I13* referenced in the formula. In the template, the range includes just a single cell. However, when populating the template with data, ExcelWriter automatically expands the range to encompass all the rows inserted for the group. Consequently, the *SUBTOTAL* formula will operate over the correct range of values for each group.

To generate data for the chart, we insert a second worksheet. To show just the grouped value for each order year, we hide the data marker row by right-clicking on the row header then selecting *Hide* (see figure). The data for the chart should span the area between and including the *%%group* and *%%endgroup* markers. To unhide a row in Excel 2007, type Control-G and enter a cell address then select *Format* > *Hide & Unhide* > *Unhide Rows* on the Home ribbon.



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
```

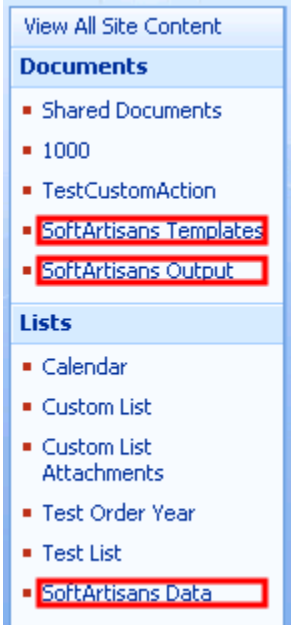
To retract the SharePoint solution package:

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

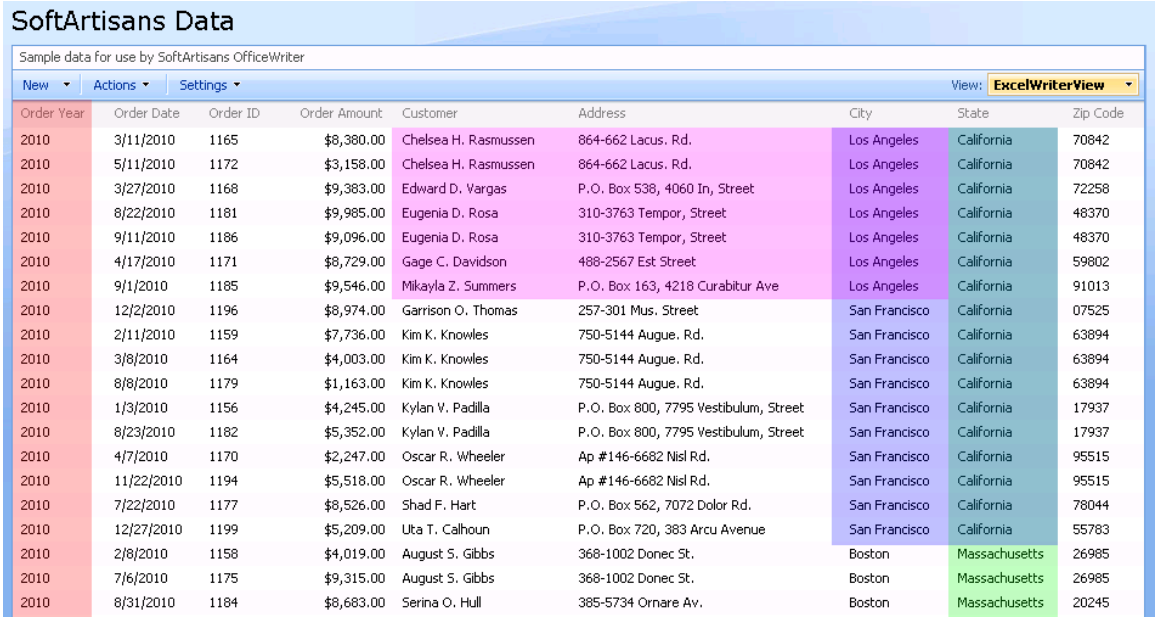
Step 10: Using the tutorial

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

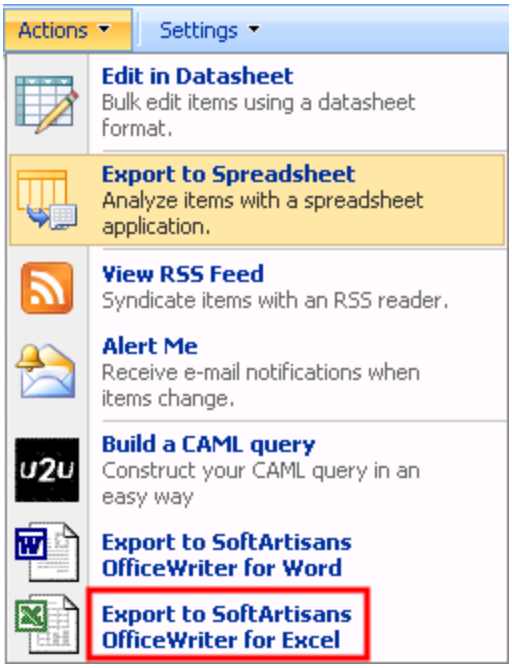


Open the *SoftArtisans Data* list. Select *ExcelWriterView* from the View drop-down. The data are sorted by year (in descending order), by state, by city, then by customer. The sorting order is the same as that expected in the template.

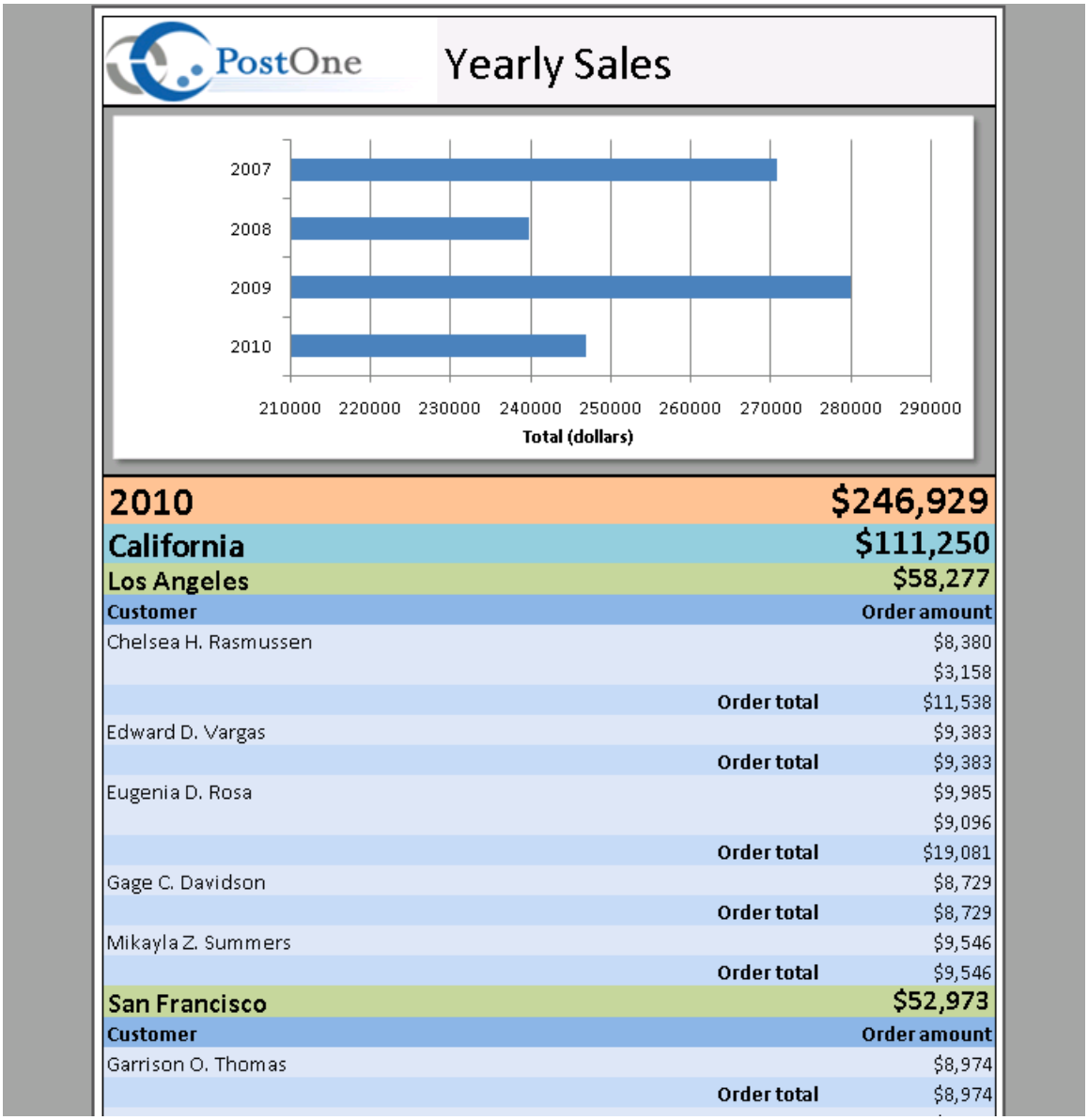


Order Year	Order Date	Order ID	Order Amount	Customer	Address	City	State	Zip Code
2010	3/11/2010	1165	$8,380.00	Chelsea H. Rasmussen	864-662 Lacus. Rd.	Los Angeles	California	70842
2010	5/11/2010	1172	$3,158.00	Chelsea H. Rasmussen	864-662 Lacus. Rd.	Los Angeles	California	70842
2010	3/27/2010	1168	$9,383.00	Edward D. Vargas	P.O. Box 538, 4060 In, Street	Los Angeles	California	72258
2010	8/22/2010	1181	$9,985.00	Eugenia D. Rosa	310-3763 Tempor, Street	Los Angeles	California	48370
2010	9/11/2010	1186	$9,096.00	Eugenia D. Rosa	310-3763 Tempor, Street	Los Angeles	California	48370
2010	4/17/2010	1171	$8,729.00	Gage C. Davidson	488-2567 Est Street	Los Angeles	California	59802
2010	9/1/2010	1185	$9,546.00	Mikayla Z. Summers	P.O. Box 163, 4218 Curabitur Ave	Los Angeles	California	91013
2010	12/2/2010	1196	$8,974.00	Garrison O. Thomas	257-301 Mus. Street	San Francisco	California	07525
2010	2/11/2010	1159	$7,736.00	Kim K. Knowles	750-5144 Augue. Rd.	San Francisco	California	63894
2010	3/8/2010	1164	$4,003.00	Kim K. Knowles	750-5144 Augue. Rd.	San Francisco	California	63894
2010	8/8/2010	1179	$1,163.00	Kim K. Knowles	750-5144 Augue. Rd.	San Francisco	California	63894
2010	1/3/2010	1156	$4,245.00	Kylan V. Padilla	P.O. Box 800, 7795 Vestibulum, Street	San Francisco	California	17937
2010	8/23/2010	1182	$5,352.00	Kylan V. Padilla	P.O. Box 800, 7795 Vestibulum, Street	San Francisco	California	17937
2010	4/7/2010	1170	$2,247.00	Oscar R. Wheeler	Ap #146-6682 Nisl Rd.	San Francisco	California	95515
2010	11/22/2010	1194	$5,518.00	Oscar R. Wheeler	Ap #146-6682 Nisl Rd.	San Francisco	California	95515
2010	7/22/2010	1177	$8,526.00	Shad F. Hart	P.O. Box 562, 7072 Dolor Rd.	San Francisco	California	78044
2010	12/27/2010	1199	$5,209.00	Uta T. Calhoun	P.O. Box 720, 383 Arcu Avenue	San Francisco	California	55783
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
2010	8/31/2010	1184	$8,683.00	Serina O. Hull	385-5734 Ornare Av.	Boston	Massachusetts	20245

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



ExcelWriter then retrieves the template, pulls the data from the list, and populates the template. The generated spreadsheet is saved to the *SoftArtisans Output* document library and streamed to the browser. Below is a part of the output.



SharePoint Solution - Excel Export Plus

Excel Export Plus Introduction

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

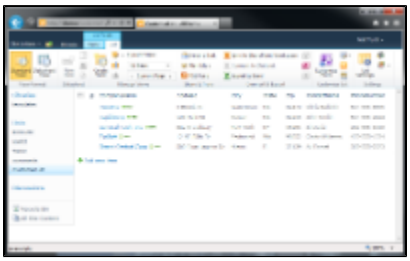
Whereas using the out-of-the-box SharePoint export-to-Excel feature allows only for raw, unformatted data dumps, using OfficeWriter and Excel Export Plus supports:

1. Custom Excel templates
2. Charts, formulas, pivot tables, or any other Excel feature
3. Exporting to XLS and XLSX files
4. With developer customization, you can exercise runtime control over just about every aspect of the workbook

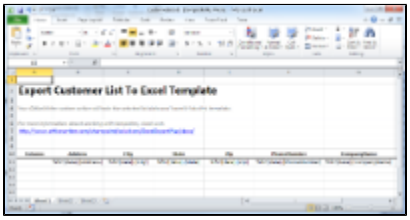
After you install and configure Excel Export Plus, you will find a button in the ribbon of your list that exports the list data into a pre-defined template in Excel. The "List to Excel" ribbon button below is an example of Excel Export Plus. You can customize the button text based on your requirements.

How it Works

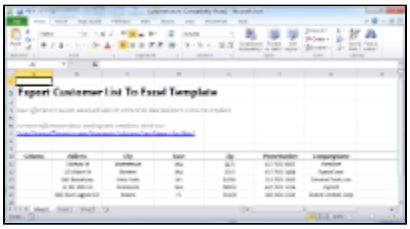
Excel Export Plus uses the OfficeWriter API under SharePoint's hood to merge SharePoint list item data into a Excel workbook. For example, say you have a list like this one, with CompanyName, Address, City, State, Zip, ContactName, and PhoneNumber columns:



You could design a template like this in Excel. Put anything you want into the Excel workbook document, and type specially formatted data markers that correspond to the names of the columns in the list. These data markers specify where the list data will be merged into the template.



When you choose to export the list data by clicking the ribbon button, Excel Export Plus pulls the values from the list and then uses OfficeWriter to open the template file, replace the data markers with actual data, and stream the populated workbook to the browser:



Fully Customizable by Developers

You can install OfficeWriter and Excel 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 Excel 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 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 Excel 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 "Excel Export Plus.wsp"
    c. Click "Deploy Solution"
    d. Specify a deployment time
    e. Click "OK"

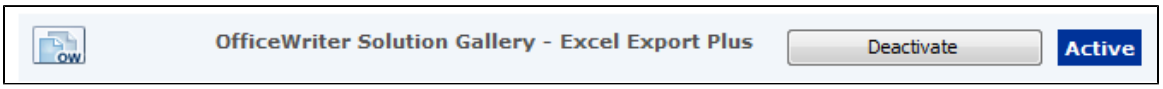
Configuration

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 Excel Export Plus:

#Browse to the SharePoint site in which you want to use this solution.

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



Configuration

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

1. From "Site Settings", click "OfficeWriter Solution: Excel 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 Excel Export Plus.
3. In the "Excel Template" section, select "Create a new template file" and then choose the columns from the list that you want to appear in the export. Excel Export Plus will automatically create an Excel template for you with the data markers that you choose (you can go back later and customize the template in Excel). You can also select a previously created template from any library in the SharePoint site.
4. If creating a new template, specify a document library in which you'd like it to be stored.
5. Type a name for the new template file
6. Finally, in the "Action Name" section specify the text you want to appear on ribbon button, and provide an optional description.

Here is an example 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 Excel template files, see here:
http://wiki.softartisans.com/display/EW70/How+to+Use+Templates

Installation

- Installing ExcelWriter
- Removing ExcelWriter
- Security Considerations

Installing ExcelWriter

- Upgrading to a New Version of ExcelWriter
- Automatic Installation
- ExcelWriter Installation Files
- Installing ExcelWriter at the Application Level
- Installing ExcelWriter in the Global Assembly Cache (GAC)
- Using ExcelWriter with Windows 2003

Upgrading to a New Version of ExcelWriter

To replace an installed version of ExcelWriter with version 7, remove the old version and then run the automatic installer. If you want to run version 7 side-by-side with a previous version of ExcelWriter, run the installer on another computer, and copy Softartisans.OfficeWriter.ExcelWriter.dll and LicenseManager.exe to the target machine. Then, run LicenseManager.exe and enter your new license key.

Automatic Installation

To run the automatic installation, double-click OfficeWriter-4.x.x-Win.exe and follow the instructions. By default, the automatic installation will save ExcelWriter's assemblies in the directory C:\Program Files\SoftArtisans\OfficeWriter\bin\dotnet20. You can select a different directory during the installation process.

> ⚠ ExcelWriter COM is no longer included in the standard install for OfficeWriter. If you need ExcelWriter COM, please contact SoftArtisans directly.

During the automatic installation process you will be asked to enter the ExcelWriter 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 License Key Manager (Start > Programs > SoftArtisans > OfficeWriter 4.x Windows > License Key Manager).

When you install ExcelWriter, you will also be given the option to install WordWriter. If your license key is only for ExcelWriter, and you choose to install WordWriter as well, a free limited version of WordWriter 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.

ExcelWriter Installation Files

File	Description
Softartisans.OfficeWriter.ExcelWriter.dll	Softartisans.OfficeWriter.ExcelWriter.dll contains: • The pure .NET ExcelTemplate object. • The pure .NET ExcelApplication object. * To run an ExcelWriter application in ASP.NET, copy Softartisans.OfficeWriter.ExcelWriter.dll to the application's \bin directory or to the Global Assemble Cache (GAC).
LicenseManager.exe	The License Manager allows you to check what OfficeWriter license keys are installed in your system registry, and to add or remove license keys (when upgrading, for example).

OWAssist.cab *	OWAssist.cab contains OfficeWriter Assistant, a file transfer tool included with ExcelWriter Enterprise Edition.

* This feature is not available in ExcelWriterSE.

Installing ExcelWriter at the Application Level

To make ExcelWriter available to an application, include a directory called **bin** at the application's top level, and save a copy of Softartisans.OfficeWriter.ExcelWriter.dll in this directory:

1. At the top level of your application, create a directory called **bin**.
2. Copy Softartisans.OfficeWriter.ExcelWriter.dll from OfficeWriter\ExcelWriter\ to your application's bin directory.

If you are using Visual Studio .NET, add the assembly to your project:

1. Right-click the project name in Solution Explorer and choose Add Reference.
2. Click Browse and navigate to C:\Program Files\SoftArtisans\OfficeWriter\bin.
3. Choose Softartisans.OfficeWriter.ExcelWriter.dll and click Open. Click OK.

Installing ExcelWriter in the Global Assembly Cache (GAC)

To make ExcelWriter available to all ASP.NET applications on your machine, install ExcelWriter in the Global Assembly Cache (GAC):

1. Open a command prompt window and move to the directory OfficeWriter\bin\dotnet20\.
2. Enter gacutil /i SoftArtisans.OfficeWriter.ExcelWriter.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.ExcelWriter,
            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.ExcelWriter,
                    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 Softartisans.OfficeWriter.ExcelWriter.dll added to the GAC. If you add a new version of Softartisans.OfficeWriter.ExcelWriter.dll to the GAC using the gacutil command, update the dll version attributes in machine.config. To get the exact version of Softartisans.OfficeWriter.ExcelWriter.dll, right-click the file and select the Properties tab.

Using ExcelWriter with Windows 2003

To use ExcelWriter on Windows 2003, you must enable the following Web Service extensions in IIS:

- ASP.NET
- Active Server Pages (only required for ASP samples)

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 **ASP.NET** .
5. Click the **Allow** button.
6. If you are using ASP samples, select **Active Server Pages** rom the list of Web Service Extensions.
7. Click the **Allow** button.

Using ExcelWriter with Windows 2008

To use the ExcelWriter ASP samples on Windows 2008 or Windows Vista, you must install ASP on your server, as it may not be installed by default.

To install ASP on a Windows 2008 or Windows Vista server:

1. Open Programs and Features.
2. Select the **Turn Windows features on or off** link from the bar on the left.
3. Navigate to Internet Information Services > World Wide Web Services > Application Development Features and make sure **ASP** is checked.
4. Press **OK**.

Removing ExcelWriter

Automatic Removal

If ExcelWriter was installed with the automatic installer, use **Add/Remove Programs** to remove ExcelWriter 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 ExcelWriter**.
4. Click the **Change/Remove** button to remove the old copy of ExcelWriter.

If ExcelWriter was manually installed, follow the manual removal instructions.

Manual Removal

To manually remove ExcelWriter:

1. Remove the SoftArtisans.OfficeWriter.ExcelWriter.dll from the Global Assembly Cache:

   > Installation in the GAC is optional. If SoftArtisans.OfficeWriter.ExcelWriter.dll is not in the GAC, go to step 2.

   a. Open a DOS prompt window.

   b. Go to the folder OfficeWriter\ExcelWriter\bin.

   c. Enter gacutil.exe /u SoftArtisans.OfficeWriter.ExcelWriter and click Ok.

   d. Delete the following line from machine.config:

   ```
   <add assembly="SoftArtisans.OfficeWriter.ExcelWriter, Version=x.x.x.x, Culture=neutral,
   PublicKeyToken=f593502af6ee46ae"/>
   ```

2. Remove ExcelWriter's license key:

   a. Double-click **LicenseManager.exe**.

   b. Select ExcelWriter's license key.

   c. Click **Remove Key** .

3. Delete all copies of SoftArtisans.OfficeWriter.ExcelWriter.dll.

Security Considerations

ExcelWriter executes in one of the following user contexts:

- The NETWORK SERVICE account (IIS6 and IIS7)
- The ASPNET account (IIS5)
- The authenticated user account
- The impersonated user account (when impersonation is turned on)

When an ExcelWriter application opens an existing spreadsheet - either using ExcelTemplate or ExcelApplication.Open - the application's security context (NETWORK SERVICE, ASPNET, authenticated user, or impersonated user) must have **Read** access to this spreadsheet.

If you want to save a generated Excel file on the server (rather than stream it to the 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.

What is HotCell Technology?

ExcelWriter's HotCell Technology provides the ability to dynamically retrieve or update data on a Web server based on modifications made in an Excel file on a client. HotCell Technology uses Excel VBA code embedded in the spreadsheet on the client. The VBA code communicates with an application on a remote Web server. The VBA code is activated when the user modifies a cell's value. A "HotCell" is a cell in the client-side spreadsheet that is modified by the user.

There are two different upload methods that you can use to submit the changed cell data back to the server:

- **Upload the entire workbook to the server**

  This method uploads the entire edited workbook back to the server. When you issue the command to update the data source, OfficeWriter Assistant (included in OWAssis.cab) uploads the entire workbook back to the server along with information about which cells have changed. On the server, ExcelWriter opens the uploaded workbook, reads the values of the changed cells, and updates the data source. For more information about OfficeWriter Assistant, see the OfficeWriter Assistant documentation.

- **POST changed-cell instructions to the server**

  This method submits cell information in an HTTP POST request. On the server, the receiving page parses the information and updates the data source. This solution is far less network-intensive because it just posts bits of information instead of the entire workbook. However, the POST method is more complicated than the upload method and requires more code on both the client and server.

HotCell Samples

- **HotCell Web Form Sample (C#)**
  Hot Cell Web Form Sample (VB.NET)

  This sample downloads a workbook of employee information populated by ExcelTemplate. From within Excel, you can edit employee address information and update the server-side database.

- **HotCell Drill Down Sample (C#)**
  HotCell Drill Down Sample (VB.NET)

  This is a "Drill Down" report that allows you to drill through several layers of relational data. The application starts on a list of customers from the AdventureWorks2000 database. Select a customer to "drill down" and see a list of all orders made for that customer. Select an order to see the order details for that particular order.

Programmer's Reference

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

- ExcelTemplate

  Introduction to ExcelTemplate:

  Create a Spreadsheet with ExcelTemplate

  ExcelTemplate is the main class for template-driven document generation. This object opens an ExcelWriter template file, populates it with data from a specified data source, and generates a new Excel workbook.

- ExcelApplication

  > ⚠ ExcelApplication is not available in ExcelWriterSE.

  Introduction to ExcelApplication: Create a Spreadsheet with ExcelApplication

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

- SharePointIntegration

  > ⚠ SharePointIntegration is not available in ExcelWriterSE.

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

ExcelTemplate provides an intuitive high-performance way to import database values to a spreadsheet, but cannot otherwise modify a spreadsheet at runtime. ExcelApplication's rich object model allows you to modify every aspect of the spreadsheet at runtime. You can take advantage of the features of both ExcelApplication and ExcelTemplate by using them together. For example, you can use ExcelTemplate to open and populate an ExcelWriter template, then pass the populated workbook to ExcelApplication and add a chart. Or, you could generate a template spreadsheet with ExcelApplication and pass the spreadsheet to ExcelTemplate to be populated.

ExcelApplication Programmer's Reference

- Anchor — An anchor represents the position of a floating (non-cell data) object within a spreadsheet. The position is maintained as a row number + a percentage of the row's width and a column number + a percentage of the column's height.
- Area — An Area object represents a rectangular set of cells.
- Axis — An Axis object represents a chart axis. The `Axis` class has two sub-classes: CategoryAxis and ValueAxis.
- AxisLabel — An AxisLabel object represents the labels on an axis. The labels are the displayed values on the axis. Note: the title of an axis (returned by Axis.Title) is not an axis label.
- AxisType — When creating a chart with two X or Y axes (primary and secondary), use AxisType to associate a series with a particular axis.
- Border — A Border object represents a border around a cell or area of cells.
- BorderPart — A BorderPart object represents a particular part of a border around a cell or area of cells.
- CategoryAxis — A CategoryAxis object represents a chart's category (x) axis. `CategoryAxis` is a sub-class of Axis.
- Cell — A Cell object represents a single cell in a worksheet. To return a `Cell` object use <dotnet>Worksheet.Cells. Specify the cell by 0-based row and column indexes or by Excel-style reference:
- Cells — A Cells collection represents the set of cells in a worksheet.
- CellStyle — The unique cell style for a given cell.
- CharacterRun — The CharacterRun class allows you to set a font on a specified range of characters within a ChartText object or a cell. To return a `CharacterRun` object, use Cell.GetCharacters() or ChartText.GetCharacters().
- Chart — The Chart class represents an Excel chart. It is the main class for code-based chart creation and modification.
- ChartDataTable — A ChartDataTable object represents a chart's data table. A data table is a grid that can be added to some charts that contains the numeric data used to create the chart. The data table is usually attached to the chart's category axis, replacing the tick-mark labels on the axis.
- ChartFillPattern
- ChartInterior — A ChartInterior object contains formatting properties of a ChartRegion or an object extending `ChartRegion` (AxisLabel, ChartText, Legend, or SeriesDataLabel).
- ChartLine — A ChartLine object represents a line in a chart.
- ChartPageSetup — The ChartPageSetup class represents the page layout properties of a printed chart or chart sheet. To return a `ChartPageSetup` object, use Chart.PageSetup
- ChartRegion — A ChartRegion object represents either a chart area or plot area. The following classes extend `ChartRegion` : AxisLabel, ChartText, Legend, DataPoint, and SeriesDataLabel.
- Charts — The Charts collection contains all charts in the specified worksheet. Chart sheets are not included in this collection. To return a `Charts` collection, use Worksheet.Charts.
- ChartSettings3D — The ChartSettings3D class contains the set of properties specific to three-dimensional charts. To return a `ChartSettings3D` object, use Chart.ChartSettings3D.
- Chartsheet — A Chartsheet object represents a worksheet that contains only a chart.

- ChartText — A ChartText object represents a region in a chart where text is displayed (e.g., a chart title). The `ChartText` extends ChartRegion.
- ChartTextCollection — The ChartTextCollection contains all ChartText objects in the specified chart..
- ChartType — The ChartType class contains all available chart types. When setting a chart type, always specify both the major type and the sub-type, for example:
- Color — The {msdn:System.Drawing.Color|Color} class represents a single color in a spreadsheet's color palette.
- ColumnProperties — The ColumnProperties class represents a single column in a worksheet.
- Comment — A Comment object represents a note attached to a cell, or group of merged cells, separate from other cell content. A `Comment` must be associated with a cell, and may only be created using set Comment.Visible to `true`.
- Comments — The Comments collection contains all Comment objects in the worksheet. To return a `Comments` collection, use Worksheet.Comments.
- Condition — A Condition object represents a condition within a ConditionalFormat, a format that Excel will apply to a cell or group of cells if a specified condition is met.
- ConditionalFormat — A ConditionalFormat object represents a conditional format that Excel will apply to a cell or group of cells if a specified condition is met. Each `ConditionalFormat` object contains up to three Condition and a Range object representing the set of cells on which the conditions will be tested.
- CopyPasteProperties — The CopyPasteProperties object represents the types of data that will be copied from a worksheet when you call the CopyPaste() method.
- DataImportProperties — The DataImportProperties contains a set of properties that can be used when importing data into a Worksheet or Area. To return a `DataImportProperties` object, call Workbook.CreateDataImportProperties.
- DataLabelPlacementType — The possible positions of a data label relative to its data point.
- DataPoint — A DataPoint object represents a data point in a chart. To return a `DataPoint` object, use Series[i].
- DataPointMarker — The DataPointMarker class represents the style and formatting of data markers in line, radar, and scatter charts. To return a `DataPointMarker` object, use Series.DataPointMarker.
- DataValidation — A DataValidation object represents a rule for validating data entered in cells. The rule may be applied to a cell or a set of cells. The Workbook class contains several CreateDataValidation methods for creating `DataValidation` objects. To apply a `DataValidation` object to a cell or group of cells, use Area.SetDataValidation or Range.SetDataValidation.
- DocumentProperties — The DocumentProperties object represents properties of a spreadsheet. These properties are displayed in the file's Properties sheet which can be opened in Excel (File -> Properties), or by right-clicking the file icon and selecting "Properties." The `DocumentProperties` class can be used to remove personal information from an existing spreadsheet, or to populate searchable fields that can be used to locate the file.
- ErrorBar — An ErrorBar object represents series error bars. Error bars graphically express potential error amounts relative to each data marker in a data series. You can add error bars to data series in two-dimensional area, bar, column, line, scatter, and bubble charts. In scatter and bubble charts, you can display error bars for x values, y values, or both.
- ExcelApplication — ExcelApplication is the main class for pure code-based workbook generation (for information about template-based workbook generation, see ExcelTemplate). This class is an engine used to open, create, and write (save or stream to a browser) workbooks. A single instance of `ExcelApplication` can generate multiple workbooks.
- ExcelWriterException — All methods in the ExcelWriter API can throw an exception of type ExcelWriterException.
- Font — The Font object represents a font style that you can use when creating a reusable Style object, or when setting the style of a cell or set of cells. To set or return a `Font` object, use Style.Font.
- GlobalStyle — ExcelWriter creates three style types: GlobalStyle, NamedStyle, and CellStyle. All three style types derive from the Style class, and `NamedStyle` derives from `GlobalStyle`.
- Group — A Group is a type of shape that contains other shapes. Currently, groups may not be created or have their contents modified. They may be moved, resized, and deleted.
- Groups — Groups provides access to the collection of Group objectss contained within a drawing. Currently, only top level groups may be accessed. Nested groups are not supported. Additionally, a group may not be created or have its contents modified. However, a group can be moved, resized, and removed from a drawing. To return a `Groups` collection, use Worksheet.ShapeGroups.
- HeaderFooterPicture — A HeaderFooterPicture object represents a picture embedded in a Header or Footer.
- HeaderFooterSection — The HeaderFooterSection class represents an area of the header or footer. It allows you to set and retrieve the content in that area.
- Hyperlink — Represents a hyperlink in an Area (which may contain only one cell). To create a Hyperlink object, call Cell.CreateHyperlink or Area.CreateHyperlink. To get an existing `Hyperlink`, use
- Hyperlinks — The Hyperlinks collection contains all hyperlinks in a worksheet. The Worksheet.Hyperlinks property returns a `Hyperlinks` collection.
- Legend — A Legend object represents a chart's legend. To return `Legend` object, use Chart.Legend.
- NamedObject — A `NamedObject` object may refer to an array, number, picture, or range. To create a `NamedObject` in an Excel file, use the corresponding method for the specific object you wish to create (Pictures.CreatePicture(), Workbook.CreateRange(), etc). To get a `NamedObject` from an existing file, use Workbook.GetNamedObject(), passing in the name of the object you wish to get.
- NamedStyle — ExcelWriter creates three style types: GlobalStyle, NamedStyle, and CellStyle. All three style types derive from the Style class, and `NamedStyle` derives from `GlobalStyle`.
- NumberFormat — NumberFormat is a helper class that creates custom number format strings, which are used for formatting numbers and dates. Use Style.NumberFormat to set a number format for a specific style.
- ObjectPositioning — Enumeration of the available resize and reposition behaviors of an object when the cells in the spreadsheet it spans are changed.
- Othersheet — A Othersheet object represents a sheet that is not a worksheet or a chartsheet. This sheet represents VBA Module and dialog sheets. These are not currently implemented by ExcelWriter
- PageSetup — The PageSetup class represents the page layout properties of a printed worksheet.
- Palette — The Palette class represents Excel's color palette.
- Pattern — Pattern contains all available background patterns for charts and pictures.
- Picture — A Picture object represents a picture in a worksheet and allows the size, position, and content of the picture to be managed. Once created, the data of a picture may not be changed or replaced with the data of another picture. Instead, delete the picture and create a new one.
- Pictures — The Pictures collection represents the set of pictures in a worksheet. To return a `Pictures` collection, use Worksheet.Pictures.

- Range — A Range object represents a range in a workbook.
- RowProperties — The RowProperties class represents a single row in a worksheet. To return a `RowProperties` object, use Worksheet.GetRowProperties.
- Series — A Series object represents a single data series in a chart. To return a `Series` object, use one of the SeriesCollection.CreateSeries methods.
- SeriesCollection — The SeriesCollection class contains the set of data series in a worksheet.
- SeriesDataLabel — A SeriesDataLabel represents the appearance of series labels.
- SettingsBarColumn — The SettingsBarColumn class contains all settings that are specific to bar and column charts. To return a `SettingsBarColumn` object,
- SettingsLineAreaScatter — The SettingsLineAreaScatter class contains all settings that are specific to line, area, and scatter charts. To return a `SettingsLineAreaScatter` object,
- SettingsPieDoughnut — The SettingsPieDoughnut class contains all settings that are specific to pie and doughnut charts. To return a `SettingsPieDoughnut` object,
- Shape — The Shape class allows the appearance and behavior of drawing object to be manipulated. On its own, `Shape` represents a simple auto shape, such as a line, rectangle, or oval. A shape may also represent the display area of a Comment or the size, position, and formatting of a Picture.
- Shapes — The Shapes collection contains all Shape objects in the worksheet. To return a `Shapes` collection, use Worksheet.Shapes.
- ShapeType — The ShapeType class contains the Excel shape types supported by ExcelWriter.
- Style — ExcelWriter has three style types: CellStyle, GlobalStyle, and NamedStyle.
- Trendline — A Trendline object represents a series trendline. To create a `Trendline` object, call Trendlines.CreateTrendline .
- Trendlines — Trendlines is a collection of all Trendline objects associated with a data series in a chart. To return a `Trendlines` collection, use Series.Trendlines .
- UpDownBar — An UpDownBar object represents up/down bars in a line chart. Up/down bars connect data points in the highest and lowest series in a line chart to show motion during an interval.
- ValueAxis — A ValueAxis object represents a chart's value (y) axis. The `ValueAxis` class extends Axis .
- Workbook — A Workbook object represents an Excel workbook. To return a `Workbook` object: Call ExcelApplication.Create() to create a new Excel workbook. Call ExcelApplication.Open() to open an existing Excel workbook.
- Worksheet — The Worksheet class represents a single Excel worksheet.
- Worksheets — The Worksheets class represents the set of worksheets in an Excel workbook. This object is used to manage (add, copy, remove, etc.) the worksheets in the workbook To return a `Worksheets` object, use Workbook.Worksheets.

Anchor

Description

An anchor represents the position of a floating (non-cell data) object within a spreadsheet. The position is maintained as a row number + a percentage of the row's width and a column number + a percentage of the column's height.

Signature: C#

```
public sealed class Anchor
```

Signature: VB.NET

```
Public NotInheritable Class Anchor
```

Remarks

To create an Anchor object, use Worksheet.CreateAnchor or Cell.CreateAnchor.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Anchor anc = ws.CreateAnchor(2, 4, 0, 0);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim anc As Anchor = ws.CreateAnchor(2, 4, 0, 0)
```

Properties

Name	Description
Column	Returns the 0-based number of the column containing the anchor.
OffsetX	Returns the X offset from the left side of the column containing the anchor, as a percentage of column width.
OffsetY	Returns the Y offset from the top of the row containing the anchor, as a percentage of row height.
Row	Returns the 0-based number of the row containing the anchor..

Anchor.Column

Description

Returns the 0-based number of the column containing the anchor.

Signature: C#

```
public int Column{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Column() As Integer
```

Examples

C#

```
int col = anc.Column;
```

vb.net

```
Dim col As Integer = anc.Column
```

Anchor.OffsetX

Description

Returns the X offset from the left side of the column containing the anchor, as a percentage of column width.

Signature: C#

```
public double OffsetX{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property OffsetX() As Double
```

Remarks

An offset of `100.0` represents an anchor flush with the right side of the column.

Examples

C#

```
double x = anc.OffsetX;
```

vb.net

```
Dim x As Double = anc.OffsetX
```

Anchor.OffsetY

Description

Returns the Y offset from the top of the row containing the anchor, as a percentage of row height.

Signature: C#

```
public double OffsetY{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property OffsetY() As Double
```

Remarks

An offset of 100.0 represents an anchor flush with the bottom side of the row.

Examples

C#

```
double y = anc.OffsetY;
```

vb.net

```
Dim y As Double = anc.OffsetY
```

Anchor.Row

Description

Returns the 0-based number of the row containing the anchor..

Signature: C#

```
public int Row{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Row() As Integer
```

Examples

C#

```
int row = anc.Row;
```

vb.net

```
Dim row As Integer = anc.Row
```

Area

Description

An Area object represents a rectangular set of cells.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Area
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Area
```

Remarks

To create an `Area` object, call Worksheet.CreateArea

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Area a = ws.CreateArea(4, 4, 15, 6);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb(0)
          Dim a As Area = ws.CreateArea(4, 4, 15, 6)
```

Properties

Name	Description
AllColumnWidths	Sets the width - in points (1/72 of an inch) - of all columns in the Area. When converted to character widths, this must be a value that is between 0 - 255.
AllColumnWidthsInChars	Sets or returns the width of all columns in this Area as a number of '0' characters in the font of the 'Normal' style. This must be a value between 0 - 255.
AllRowsHeight	Sets the height - in points (1/72 of an inch) - of all rows in this Area. This must be a value that is between 0 - 409.
ArrayFormula	Sets or returns an array entered formula. An array entered formula is a formula that can have multiple output values. To set an array entered formula, surround the formula string with a '{' and '}'.

BorderAround	Returns a Border object that represents a border around the area. Use the `Border` object returned to set the appearance of the border.
ColumnCount	Sets or returns the number of columns in the area.
Dimensions	Sets or returns the dimensions of the area in the format "A1:B7".
FirstColumn	Sets or returns the 0-based number of the first column within the rectangular area of cells.
FirstRow	Sets or returns the 0-based number of the first row within the rectangular area of cells.
PopulatedCells	Returns a rectanglular area of cells from the first (top, left) populated cell to the last (bottom, right) populated cell in the Area. The Area returned will include cells that contain no data but have been formatted.
RowCount	Sets or returns the number of rows in the area.
WorksheetIndex	Returns the 0-based index of the worksheet that contains the specified area.

Indexers

Name	Description
Item(Int32, Int32)	Returns a cell within the specified area of cells.

Methods

Name	Description
ApplyStyle(Style)	Applies a style to the area. When a style is applied - rather than set (see Area.SetStyle(Style)) - only the differences between the new style and style properties previously assigned to the area (through the ExcelWriter API or in Microsoft Excel) will take effect. For example, if the area has a background color and the new style applied does not contain a background color, the area's color will not be affected. However, if the new style includes background color, it will replace the existing background color of the area.
AutoFitHeight()	Sets the height of each of the rows within the area to the height of the highest cell value in that row within the bounds of the Area.
AutoFitWidth()	Sets the width of each of the columns within the area to the width of the widest cell value in that column within the bounds of the Area.
ClearContent()	Clears the content of all cells in the area. Calling this method is equivalent to setting an area's cell values to null.
CreateHyperlink(String)	Creates a new hyperlink in the area pointing to the specified href. This hyperlink is automatically added to the worksheet. All cell values in the area will be converted to hyperlinks.
GetColumnWidth(Int32)	Returns the width - in points (1/72 of an inch) - of the specified column. When converted to character widths, this must be a value that is between 0 - 255.
GetColumnWidthInChars(Int32)	Sets or returns the width of the column as a number of '0' characters in the font of the 'Normal' style. This must be a value between 0 - 255.
GetRowHeight(Int32)	Returns the height - in points (1/72 of an inch) - of the specified row. When converted to character widths, this must be a value that is between 0 - 409.
GroupColumns(Boolean)	Groups or outlines the columns contained within the area.
GroupRows(Boolean)	Groups or outlines the rows contained within the area.
ImportData(Object()())	Imports data from a two-dimensional array of objects to the specified Area . The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object()(), String(), DataImportProperties)	Imports data from a two-dimensional array of objects to the specified Area . The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataTable)	Imports data from an ADO.NET `DataTable` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataTable, DataImportProperties)	Imports data from an ADO.NET `DataTable` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

ImportData(System.Data.DataView)	Imports data from an ADO.NET `DataView` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataView, DataImportProperties)	Imports data from an ADO.NET `DataView` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object(,))	Imports data from a rectangular array of objects to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object(,), String(), DataImportProperties)	Imports data from a rectangular array of objects to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.IDataReader)	Imports data from an `IDataReader` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.IDataReader, DataImportProperties)	Imports data from an `IDataReader` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
MergeCells()	Merges all cells in the area into a single cell.
RemoveConditionalFormat()	This method removes any ConditionalFormat objects from the Area.
SetColumnWidth(Int32, Double)	Sets the width - in points (1/72 of an inch) - of the specified column. When converted to character widths, this must be a value that is between 0 - 255.
SetColumnWidthInChars(Int32, Double)	Sets or returns the width of the column as a number of '0' characters in the font of the 'Normal' style. This must be a value between the 0 - 255.
SetConditionalFormat(ConditionalFormat)	This method copies the specified ConditionaFormat object and associates it with this Area. If any ConditionalFormat objects already exist within the Area, they will be removed and replaced by the specifed one.
SetDataValidation(DataValidation)	Assigns a data validation rule to all cells in the Area.If the dataValidation object uses a local reference or area in a formula (i.e. "=A5:B7"), then the area will adjust those local references accordingly based on the area's location. The original dataValidation object will not be effected.
SetRowHeight(Int32, Double)	Sets the height - in points (1/72 of an inch) - of the specified row. When converted to character widths, this must be a value that is between 0 - 409.
SetStyle(Style)	Sets or returns a style for the area. When a style is set - rather than applied (see Area.ApplyStyle) - the area to which the style is assigned acquires all of that style's properties, including font properties and number formatting. Setting a style overwrites any pre-existing formatting on the area.
ToString()	Returns a string representation of the Area object.
UngroupColumns()	Ungroups the columns within the area.
UngroupRows()	Ungroups the rows within the area.

Extension Methods

Introduced in build 7.5

Overload	Description
ImportData(Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint List to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Microsoft.SharePoint.SPList)	Imports data from a SharePoint List to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint View to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Imports data from a SharePoint View to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Area.AllColumnWidths

Description

Sets the width - in points (1/72 of an inch) - of all columns in the Area. When converted to character widths, this must be a value that is between 0 - 255.

Signature: C#

```
public double AllColumnWidths{ set; }
```

Signature: VB.NET

```
Public WriteOnly Property AllColumnWidths() As Double
```

Examples

C#

```
a.AllColumnWidths = 100;
```

vb.net

```
a.AllColumnWidths = 100
```

Area.AllColumnWidthsInChars

Description

Sets or returns the width of all columns in this Area as a number of '0' characters in the font of the 'Normal' style. This must be a value between 0 - 255.

Signature: C#

```
public double AllColumnWidthsInChars{ set; }
```

Signature: VB.NET

```
Public WriteOnly Property AllColumnWidthsInChars() As Double
```

Examples

C#

```
a.AllColumnWidthsInChars = 100;
```

vb.net

```
a.AllColumnWidthsInChars = 100
```

Area.AllRowsHeight

Description

Sets the height - in points (1/72 of an inch) - of all rows in this Area. This must be a value that is between 0 - 409.

Signature: C#

```
public double AllRowsHeight{ set; }
```

Signature: VB.NET

```
Public WriteOnly Property AllRowsHeight() As Double
```

Examples

C#

```
a.AllRowsHeight = 30;
```

vb.net

```
a.AllRowsHeight = 30
```

Area.ApplyStyle(Style)

Description

Applies a style to the area. When a style is applied - rather than set (see Area.SetStyle(Style)) - only the differences between the new style and style properties previously assigned to the area (through the ExcelWriter API or in Microsoft Excel) will take effect. For example, if the area has a background color and the new style applied does not contain a background color, the area's color will not be affected. However, if the new style includes background color, it will replace the existing background color of the area.

Signature: C#

```
public void ApplyStyle(Style style)
```

Signature: VB.NET

```
Public Sub ApplyStyle(ByVal style As Style)
```

Parameters

style

A Style object representing the style to apply to the cell.

Examples

C#

```
a.ApplyStyle(oStyle);
```

vb.net

```
a.ApplyStyle(oStyle)
```

Area.ArrayFormula

Description

Sets or returns an array entered formula. An array entered formula is a formula that can have multiple output values. To set an array entered formula, surround the formula string with a '{' and '}'.

Signature: C#

```
public System.String ArrayFormula{ get; set; }
```

Signature: VB.NET

```
Public Property ArrayFormula() As String
```

Examples

C#

```
//--- Get ArrayFormula
string formula = a.ArrayFormula;

//--- Set ArrayFormula
Area a = ws.CreateArea(10,0,10,1);
a.ArrayFormula = "{=TREND(A1:A10,B1:B10)}";
```

vb.net

```
'--- Get ArrayFormula
Dim formula As String = a.ArrayFormula

'--- Set ArrayFormula
Dim a As Area = ws.CreateArea(10,0,10,1)
a.ArrayFormula = "{=TREND(A1:A10,B1:B10)}"
```

Area.AutoFitHeight()

Description

Sets the height of each of the rows within the area to the height of the highest cell value in that row within the bounds of the Area.

Signature: C#

```
public void AutoFitHeight()
```

Signature: VB.NET

```
Public Sub AutoFitHeight()
```

Examples

C#

```
a.AutoFitHeight();
```

vb.net

```
a.AutoFitHeight()
```

Area.AutoFitWidth()

Description

Sets the width of each of the columns within the area to the width of the widest cell value in that column within the bounds of the Area.

Signature: C#

```
public void AutoFitWidth()
```

Signature: VB.NET

```
Public Sub AutoFitWidth()
```

Examples

C#

```
a.AutoFitWidth();
```

vb.net

```
a.AutoFitWidth()
```

Area.BorderAround

Description

Returns a Border object that represents a border around the area. Use the `Border` object returned to set the appearance of the border.

Signature: C#

```
public Border BorderAround{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property BorderAround() As Border
```

Examples

C#

```
Border bordr = a.BorderAround;
```

vb.net

```
Dim bordr As Border = a.BorderAround
```

Area.ClearContent()

Description

Clears the content of all cells in the area. Calling this method is equivalent to setting an area's cell values to null.

Signature: C#

```
public void ClearContent()
```

Signature: VB.NET

```
Public Sub ClearContent()
```

Examples

C#

```
a.ClearContent();
```

vb.net

```
a.ClearContent()
```

Area.ColumnCount

Description

Sets or returns the number of columns in the area.

Signature: C#

```
public int ColumnCount{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ColumnCount() As Integer
```

Examples

C#

```
//--- Get ColumnCount
int colCount = a.ColumnCount;

//--- Set ColumnCount
a.ColumnCount = 7;
```

vb.net

```
'--- Get ColumnCount
Dim colCount As Integer = a.ColumnCount

'--- Set ColumnCount
a.ColumnCount = 7
```

Area.CreateHyperlink(String)

Description

Creates a new hyperlink in the area pointing to the specified href. This hyperlink is automatically added to the worksheet. All cell values in the area will be converted to hyperlinks.

Signature: C#

```
public Hyperlink CreateHyperlink(System.String href)
```

Signature: VB.NET

```
Public Function CreateHyperlink(ByVal href As String) As Hyperlink
```

Parameters

href

The hyperlink's destination.

Returns

A Hyperlink object.

Examples

C#

```
Hyperlink link = a.CreateHyperlink("http://www.softartisans.com");
```

vb.net

```
Dim link As Hyperlink = a.CreateHyperlink("http://www.softartisans.com")
```

Area.Dimensions

Description

Sets or returns the dimensions of the area in the format "A1:B7".

Signature: C#

```
public System.String Dimensions{ get; }
```

Signature: VB.NET

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

Examples

C#

```
//--- Get Dimensions
string areaDimensions = a.Dimensions;

//--- Set Dimensions
a.Dimensions = "A1:B7";
```

vb.net

```
'--- Get Dimensions
Dim areaDimensions As String = a.Dimensions

'--- Set Dimensions
a.Dimensions = "A1:B7"
```

Area.FirstColumn

Description

Sets or returns the 0-based number of the first column within the rectangular area of cells.

Signature: C#

```
public int FirstColumn{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property FirstColumn() As Integer
```

Examples

C#

```
//--- Get FirstColumn
int firstCol = a.FirstColumn;

//--- Set FirstColumn
a.FirstColumn = 5;
```

vb.net

```
'--- Get FirstColumn
Dim firstCol As Integer = a.FirstColumn

'--- Set FirstColumn
a.FirstColumn = 5
```

Area.FirstRow

Description

Sets or returns the 0-based number of the first row within the rectangular area of cells.

Signature: C#

```
public int FirstRow{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property FirstRow() As Integer
```

Examples

C#

```
//--- Get FirstRow
int row = a.FirstRow;

//--- Set FirstRow
a.FirstRow = 5;
```

vb.net

```
'--- Get FirstRow
Dim row As Integer = a.FirstRow

'--- Set FirstRow
a.FirstRow = 5
```

Area.GetColumnWidth(Int32)

Description

Returns the width - in points (1/72 of an inch) - of the specified column. When converted to character widths, this must be a value that is between 0 - 255.

Signature: C#

```
public double GetColumnWidth(int columnIndex)
```

Signature: VB.NET

```
Public Function GetColumnWidth(ByVal columnIndex As Integer) As Double
```

Parameters

columnIndex

The column's 0-based index, relative to the first column in the Area.

Returns

The width of the column, in points.

Examples

C#

```
double colWidth = a.GetColumnWidth(0);
```

vb.net

```
Dim colWidth As Double = a.GetColumnWidth(0)
```

Area.GetColumnWidthInChars(Int32)

Description

Sets or returns the width of the column as a number of '0' characters in the font of the 'Normal' style. This must be a value between 0 - 255.

Signature: C#

```
public double GetColumnWidthInChars(int columnIndex)
```

Signature: VB.NET

```
Public Function GetColumnWidthInChars(ByVal columnIndex As Integer) As Double
```

Parameters

columnIndex

The column's 0-based index, relative to the starting column of the Area.

Returns

The width of the column, as a number of '0' characters.

Examples

C#

```
double colWidth = a.GetColumnWidthInChars();
```

vb.net

```
Dim colWidth As Double = a.GetColumnWidthInChars()
```

Area.GetRowHeight(Int32)

Description

Returns the height - in points (1/72 of an inch) - of the specified row. When converted to character widths, this must be a value that is between 0 - 409.

Signature: C#

```
public double GetRowHeight(int rowIndex)
```

Signature: VB.NET

```
Public Function GetRowHeight(ByVal rowIndex As Integer) As Double
```

Parameters

rowIndex

The 0-based index of the row, relative to the first row in the area.

Returns

The height of the row, in points.

Examples

C#

```
double rowHeight = a.GetRowHeight(0);
```

vb.net

```
Dim rowHeight As Double = a.GetRowHeight(0)
```

Area.GroupColumns(Boolean)

Description

Groups or outlines the columns contained within the area.

Signature: C#

```
public void GroupColumns(boolean collapsed)
```

Signature: VB.NET

```
Public Sub GroupColumns(ByVal collapsed As Boolean)
```

Parameters

collapsed

`true` if the group is collapsed, `false` if not.

Examples

C#

```
a.GroupColumns(true);
```

vb.net

```
a.GroupColumns(True)
```

Area.GroupRows(Boolean)

Description

Groups or outlines the rows contained within the area.

Signature: C#

```
public void GroupRows(boolean collapsed)
```

Signature: VB.NET

```
Public Sub GroupRows(ByVal collapsed As Boolean)
```

Parameters

collapsed

`true` if the group is collapsed, `false` if not.

Examples

C#

```
a.GroupRows(true);
```

vb.net

```
a.GroupRows(True)
```

Area.ImportData

Overload	Description
ImportData(Object()())	Imports data from a two-dimensional array of objects to the specified Area . The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object()(), String(), DataImportProperties)	Imports data from a two-dimensional array of objects to the specified Area . The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataTable)	Imports data from an ADO.NET `DataTable` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataTable, DataImportProperties)	Imports data from an ADO.NET `DataTable` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataView)	Imports data from an ADO.NET `DataView` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataView, DataImportProperties)	Imports data from an ADO.NET `DataView` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object(,))	Imports data from a rectangular array of objects to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object(,), String(), DataImportProperties)	Imports data from a rectangular array of objects to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

ImportData(System.Data.IDataReader)	Imports data from an `IDataReader` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.IDataReader, DataImportProperties)	Imports data from an `IDataReader` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

SharePoint Integration Extension Methods

Overload	Description
ImportData(Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint List to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Microsoft.SharePoint.SPList)	Imports data from a SharePoint List to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint View to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Imports data from a SharePoint View to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Area.ImportData(Object(,), String(), DataImportProperties)

Description

Imports data from a rectangular array of objects to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area 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 Area
```

Parameters

data

A rectangular array of values to import to the worksheet. The first dimension corresponds to row and the second to column.
Thus, an array of data { {"A","X"},{"B","Y"},{"C","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

If you enable DataImportProperties.Transpose , the format will be [row][column], so:
{ {"A","B","C"},{"X","Y","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

columnNames

A string array of column names.

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
DataImportProperties importProps = wb.CreateDataImportProperties();
importProps.Transpose = true;
Area importedArea = a.ImportData(dataArray, fieldNames, importProps);
```

vb.net

```
Dim importProps As DataImportProperties = wb.CreateDataImportProperties()
importProps.Transpose = True
Dim importedArea As Area = a.ImportData(dataArray, fieldNames, importProps)
```

Area.ImportData(Object(,))

Description

Imports data from a rectangular array of objects to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Object[,] data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object(,)) As Area
```

Parameters

data

A rectangular array of values to import to the worksheet. The first dimension corresponds to row and the second to column.
Thus, an array of data { {"A","X"},{"B","Y"},{"C","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = a.ImportData(dataArray);
```

vb.net

```
Dim importedArea As Area = a.ImportData(dataArray)
```

Area.ImportData(Object()(), String(), DataImportProperties)

Description

Imports data from a two-dimensional array of objects to the specified Area . The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Object[][] data, System.String[] columnNames, DataImportProperties
properties)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object()(), ByVal columnNames As String(), ByVal properties
As DataImportProperties) As Area
```

Parameters

data

A two-dimensional array of values to import to the worksheet. By default, the first dimension corresponds to row and the second to column (that is, Object[row][column]).
Thus, an array of data { {"A","X"},{"B","Y"},{"C","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

If you enable DataImportProperties.Transpose , the format will be [row][column], so:
{ {"A","B","C"},{"X","Y","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

columnNames

An array of column names for the imported data. `columnNames` and the column dimension of `data` must contain the same number of elements.

properties

A DataImportProperties object that contains a set of properties to use when importing data to the area.

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
DataImportProperties importProps = oWorkbook.CreateDataImportProperties();
importProps.Truncate = true;
Area importedArea = a.ImportData(dataArray, fieldNames, importProps);
```

vb.net

```
Dim importProps As DataImportProperties = wb.CreateDataImportProperties()
importProps.Truncate = True
Dim importedArea As Area = a.ImportData(dataArray, fieldNames, importProps)
```

Area.ImportData(Object()())

Description

Imports data from a two-dimensional array of objects to the specified Area . The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Object[][] data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object()()) As Area
```

Parameters

data

A two-dimensional array of values to import to the worksheet. The first dimension corresponds to row and the second to column (that is, Object[row][column]).
Thus, an array of data { {"A","X"},{"B","Y"},{"C","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
Area data = a.ImportData(dataArray);
```

vb.net

```
Dim data As Area = a.ImportData(dataArray)
```

Area.ImportData(System.Data.DataTable, DataImportProperties)

Description

Imports data from an ADO.NET `DataTable` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.DataTable data, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataTable, ByVal props As DataImportProperties)
As Area
```

Parameters

data

The `DataTable` to import.

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
DataImportProperties importProps = wb.CreateDataImportProperties();
importProps.Truncate = true;
Area importedArea = a.ImportData(dt, importProps);
```

vb.net

```
Dim importProps As DataImportProperties = wb.CreateDataImportProperties()
importProps.Truncate = True
Area importedArea = a.ImportData(dt, importProps)
```

Area.ImportData(System.Data.DataTable)

Description

Imports data from an ADO.NET `DataTable` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.DataTable data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataTable) As Area
```

Parameters

data

The `DataTable` to import.

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = a.ImportData(dt);
```

vb.net

```
Dim importedArea As Area = a.ImportData(dt)
```

Area.ImportData(System.Data.DataView, DataImportProperties)

Description

Imports data from an ADO.NET `DataView` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.DataView data, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataView, ByVal props As DataImportProperties) As
Area
```

Parameters

data

The `DataView` to import.

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
DataImportProperties importProps = wb.CreateDataImportProperties();
importProps.Truncate = true;
Area importedArea = a.ImportData(dv, importProps);
```

vb.net

```
Dim importProps As DataImportProperties = wb.CreateDataImportProperties()
importProps.Truncate = True
Dim importedArea As Area = a.ImportData(dv, importProps)
```

Area.ImportData(System.Data.DataView)

Description

Imports data from an ADO.NET `DataView` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.DataView data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataView) As Area
```

Parameters

data

The `DataView` to import.

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = a.ImportData(oDataView);
```

vb.net

```
Dim importedArea As Area = a.ImportData(oDataView)
```

Area.ImportData(System.Data.IDataReader, DataImportProperties)

Description

Imports data from an `IDataReader` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.IDataReader data, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.IDataReader, ByVal props As DataImportProperties)
As Area
```

Parameters

data

An instance of `System.Data.SqlClient.SqlDataReader`, `System.Data.OleDb.OleDbDataReader` or

```
Microsoft.AnalysisServices.AdomdClient.AdomdDataReader
```

.
props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
DataImportProperties importProps = wb.CreateDataImportProperties();
importProps.Truncate = true;
Area importedArea = a.ImportData(dr, importProps);
```

vb.net

```
Dim importProps As DataImportProperties = wb.CreateDataImportProperties()
importProps.Truncate = True
Dim importedArea As Area = a.ImportData(dr, importProps)
```

Area.ImportData(System.Data.IDataReader)

Description

Imports data from an `IDataReader` to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.IDataReader data)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.IDataReader) As Area
```

Parameters

data

An instance of `System.Data.SqlClient.SqlDataReader`, `System.Data.OleDb.OleDbDataReader` or

```
Microsoft.AnalysisServices.AdomdClient.AdomdDataReader
```

.

Returns

An `Area` object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = a.ImportData(dr);
```

vb.net

```
Dim importedArea As Area = a.ImportData(dr)
```

Area.Item(Int32, Int32)

Description

Returns a cell within the specified area of cells.

Signature: C#

```
public Cell this[int row, int col] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal row As Integer, ByVal col As Integer) As Cell
```

Parameters

row

The cell's 0-based row number, relative to the first cell in the area. For example, for the area B4:E7, `area[0,0]` will return cell B4.

col

The cell's 0-based column number, relative to the first cell in the area. For example, for the area B4:E7, `area[0,0]` will return cell B4.

Returns

A Cell object representing the cell. If the cell is not within the specified area, an exception will be thrown.

Examples

C#

```
Cell c = ar[5,5];
```

vb.net

```
Dim c As Cell = ar(5, 5)
```

Area.MergeCells()

Description

Merges all cells in the area into a single cell.

Signature: C#

```
public void MergeCells()
```

Signature: VB.NET

```
Public Sub MergeCells()
```

Examples

C#

```
a.MergeCells();
```

vb.net

```
a.MergeCells()
```

Area.PopulatedCells

Description

Returns a rectanglular area of cells from the first (top, left) populated cell to the last (bottom, right) populated cell in the Area. The Area returned will include cells that contain no data but have been formatted.

Signature: C#

```
public Area PopulatedCells{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PopulatedCells() As Area
```

Examples

C#

```
Area populated = a.PopulatedCells;
```

vb.net

```
Dim populated As Area = a.PopulatedCells
```

Area.RemoveConditionalFormat()

Description

This method removes any ConditionalFormat objects from the Area.

Signature: C#

```
public void RemoveConditionalFormat()
```

Signature: VB.NET

```
Public Sub RemoveConditionalFormat()
```

Examples

C#

```
area.RemoveConditionalFormat();
```

vb.net

```
area.RemoveConditionalFormat()
```

Area.RowCount

Description

Sets or returns the number of rows in the area.

Signature: C#

```
public int RowCount{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property RowCount() As Integer
```

Examples

C#

```
//--- Get FirstCount
int rows = a.RowCount;

//--- Set FirstCount
a.RowCount = 7;
```

vb.net

```
'--- Get FirstCount
Dim rows As Integer = a.RowCount

'--- Set FirstCount
a.RowCount = 7
```

Area.SetColumnWidth(Int32, Double)

Description

Sets the width - in points (1/72 of an inch) - of the specified column. When converted to character widths, this must be a value that is between 0 - 255.

Signature: C#

```
public void SetColumnWidth(int columnIndex, double width)
```

Signature: VB.NET

```
Public Sub SetColumnWidth(ByVal columnIndex As Integer, ByVal width As Double)
```

Parameters

columnIndex

The column's 0-based index, relative to the first column in the area.

width

The width of the column, in points.

Examples

C#

```
a.SetColumnWidth(0, 100);
```

vb.net

```
a.SetColumnWidth(0, 100)
```

Area.SetColumnWidthInChars(Int32, Double)

Description

Sets or returns the width of the column as a number of '0' characters in the font of the 'Normal' style. This must be a value between the 0 - 255.

Signature: C#

```
public void SetColumnWidthInChars(int columnIndex, double width)
```

Signature: VB.NET

```
Public Sub SetColumnWidthInChars(ByVal columnIndex As Integer, ByVal width As Double)
```

Parameters

columnIndex

The column's 0-based index, relative to the starting column of the Area.

width

The width of the column, as a number of '0' characters.

Examples

C#

```
a.SetColumnWidthInChars(0, 100);
```

vb.net

```
a.SetColumnWidthInChars(0, 100)
```

Area.SetConditionalFormat(ConditionalFormat)

Description

This method copies the specified ConditionaFormat object and associates it with this Area. If any ConditionalFormat objects already exist within the Area, they will be removed and replaced by the specifed one.

Signature: C#

```
public void SetConditionalFormat(ConditionalFormat conditionalFormat)
```

Signature: VB.NET

```
Public Sub SetConditionalFormat(ByVal conditionalFormat As ConditionalFormat)
```

Parameters

conditionalFormat

Examples

C#

```
area.SetConditionalFormat(condFmt);
```

vb.net

```
area.SetConditionalFormat(condFmt)
```

Area.SetDataValidation(DataValidation)

Description

Assigns a data validation rule to all cells in the Area.If the dataValidation object uses a local reference or area in a formula (i.e. "=A5:B7"), then the area will adjust those local references accordingly based on the area's location. The original dataValidation object will not be effected.

Signature: C#

```
public void SetDataValidation(DataValidation dataValidation)
```

Signature: VB.NET

```
Public Sub SetDataValidation(ByVal dataValidation As DataValidation)
```

Parameters

dataValidation

A DataValidation object
representing the data validation rule to assign to the Area.

Examples

C#

```
area.SetDataValidation(myDataValidation);
```

vb.net

```
area.SetDataValidation(myDataValidation)
```

Area.SetRowHeight(Int32, Double)

Description

Sets the height - in points (1/72 of an inch) - of the specified row. When converted to character widths, this must be a value that is between 0 - 409.

Signature: C#

```
public void SetRowHeight(int rowIndex, double height)
```

Signature: VB.NET

```
Public Sub SetRowHeight(ByVal rowIndex As Integer, ByVal height As Double)
```

Parameters

rowIndex

The 0-based index of the row, relative to the first row in the area.

height

The height of the row, in points.

Examples

C#

```
a.SetRowHeight(0, 30);
```

vb.net

```
a.SetRowHeight(0, 30)
```

Area.SetStyle(Style)

Description

Sets or returns a style for the area. When a style is set - rather than applied (see Area.ApplyStyle) - the area to which the style is assigned acquires all of that style's properties, including font properties and number formatting. Setting a style overwrites any pre-existing formatting on the

area.

Signature: C#

```
public void SetStyle(Style value)
```

Signature: VB.NET

```
Public Sub SetStyle(ByVal value As Style)
```

Parameters

style

The Style object to assign to the area.

Examples

C#

```
a.SetStyle(styl);
```

vb.net

```
a.SetStyle(styl)
```

Area.ToString()

Description

Returns a string representation of the Area object.

Signature: C#

```
public virtual System.String ToString()
```

Signature: VB.NET

```
Public Overridable Function ToString() As String
```

Returns

A string representation of the `Area` object.

Examples

C#

```
string color = area.ToString();
```

vb.net

```
Dim color As String = area.ToString()
```

Area.UngroupColumns()

Description

Ungroups the columns within the area.

Signature: C#

```
public void UngroupColumns()
```

Signature: VB.NET

```
Public Sub UngroupColumns()
```

Examples

C#
`a.UngroupColumns();`

vb.net
`a.UngroupColumns()`

Area.UngroupRows()

Description

Ungroups the rows within the area.

Signature: C#

```
public void UngroupRows()
```

Signature: VB.NET

```
Public Sub UngroupRows()
```

Examples

C#
`a.UngroupRows();`

vb.net
`a.UngroupRows()`

Area.WorksheetIndex

Description

Returns the 0-based index of the worksheet that contains the specified area.

Signature: C#

```
public int WorksheetIndex{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property WorksheetIndex() As Integer
```

Examples

C#

```
int sheetNum = a.WorksheetIndex;
```

vb.net

```
Dim sheetNum As Integer = a.WorksheetIndex
```

Axis

Description

An Axis object represents a chart axis. The `Axis` class has two sub-classes: CategoryAxis and ValueAxis.

Signature: C#

```
public class Axis
```

Signature: VB.NET

```
Public Class Axis
```

Properties

Name	Description
AxisLine	Returns a ChartLine object representing the appearance of the axis.
LabelFont	Sets or returns a Font object for the labels on the axis.
MajorGridline	Returns a ChartLine object representing the major gridlines of the axis. Gridlines extend from the tick marks on an axis across the chart's plot area.
MajorTickType	Sets or returns how axis tick marks on the major gridlines are displayed.
MinorGridline	Returns a ChartLine object representing the minor gridlines of the axis. Gridlines extend from the tick marks on an axis across the chart's plot area.
MinorTickType	Sets or returns how axis tick marks on the minor gridlines are displayed.
NumberFormat	Sets or returns the display format of axis label numbers.
TextDisplayTopToBottom	Sets or returns whether the text of the tick labels is displayed top to bottom and the letters are upright.
TextRotationAngle	Sets or returns the rotation angle of the text of the tick labels.
TickLabelPosition	Sets or returns the positioning of axis tick labels.
Title	Returns a ChartText object representing the axis title.
ValuesInReverseOrder	Sets or returns whether the axis values are displayed in reverse order.
Visible	Sets or returns whether the axis will be displayed or hidden.

Nested Classes

Name	Description
TickLabelPos	A TickLabelPos value specifies the position of axis tick labels.
TickType	The TickType class contains all available axis tick types.

Axis.AxisLine

Description

Returns a ChartLine object representing the appearance of the axis.

Signature: C#

```
public ChartLine AxisLine{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property AxisLine() As ChartLine
```

Axis.LabelFont

Description

Sets or returns a Font object for the labels on the axis.

Signature: C#

```
public Font LabelFont{ get; set; }
```

Signature: VB.NET

```
Public Property LabelFont() As Font
```

Axis.MajorGridline

Description

Returns a ChartLine object representing the major gridlines of the axis. Gridlines extend from the tick marks on an axis across the chart's plot area.

Signature: C#

```
public ChartLine MajorGridline{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property MajorGridline() As ChartLine
```

Axis.MajorTickType

Description

Sets or returns how axis tick marks on the major gridlines are displayed.

Signature: C#

```
public TickType MajorTickType{ get; set; }
```

Signature: VB.NET

```
Public Property MajorTickType() As TickType
```

Axis.MinorGridline

Description

Returns a ChartLine object representing the minor gridlines of the axis. Gridlines extend from the tick marks on an axis across the chart's plot area.

Signature: C#

```
public ChartLine MinorGridline{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property MinorGridline() As ChartLine
```

Axis.MinorTickType

Description

Sets or returns how axis tick marks on the minor gridlines are displayed.

Signature: C#

```
public TickType MinorTickType{ get; set; }
```

Signature: VB.NET

```
Public Property MinorTickType() As TickType
```

Axis.NumberFormat

Description

Sets or returns the display format of axis label numbers.

Signature: C#

```
public System.String NumberFormat{ get; set; }
```

Signature: VB.NET

```
Public Property NumberFormat() As String
```

Axis.TextDisplayTopToBottom

Description

Sets or returns whether the text of the tick labels is displayed top to bottom and the letters are upright.

Signature: C#

```
public boolean TextDisplayTopToBottom{ get; set; }
```

Signature: VB.NET

```
Public Property TextDisplayTopToBottom() As Boolean
```

Axis.TextRotationAngle

Description

Sets or returns the rotation angle of the text of the tick labels.

Signature: C#

```
public int TextRotationAngle{ get; set; }
```

Signature: VB.NET

```
Public Property TextRotationAngle() As Integer
```

Axis.TickLabelPos

Description

A TickLabelPos value specifies the position of axis tick labels.

Signature: C#

```
public sealed class TickLabelPos : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class TickLabelPos
  Inherits TypesafeEnum
```

Values

Name	Description
HighEnd	Tick labels will be displayed at the top of the chart's plot area.
Invisible	Tick labels will not be shown.
LowEnd	Tick labels will be displayed at the bottom of the chart's plot area.
NextToAxis	Tick labels will be displayed next to the axis.

Axis.TickLabelPosition

Description

Sets or returns the positioning of axis tick labels.

Signature: C#

```
public TickLabelPos TickLabelPosition{ get; set; }
```

Signature: VB.NET

```
Public Property TickLabelPosition() As TickLabelPos
```

Axis.TickType

Description

The TickType class contains all available axis tick types.

Signature: C#

```
public sealed class TickType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class TickType
  Inherits TypesafeEnum
```

Values

Name	Description
CrossAxis	Tick marks will cross the axis.
InsideLine	Tick marks will face inward.
Invisible	Tick marks will not be shown.
OutsideLine	Tick marks will face outward.

Axis.Title

Description

Returns a ChartText object representing the axis title.

Signature: C#

```
public ChartText Title{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Title() As ChartText
```

Axis.ValuesInReverseOrder

Description

Sets or returns whether the axis values are displayed in reverse order.

Signature: C#

```
public boolean ValuesInReverseOrder{ get; set; }
```

Signature: VB.NET

```
Public Property ValuesInReverseOrder() As Boolean
```

Axis.Visible

Description

Sets or returns whether the axis will be displayed or hidden.

Signature: C#

```
public boolean Visible{ get; set; }
```

Signature: VB.NET

```
Public Property Visible() As Boolean
```

AxisLabel

Description

An AxisLabel object represents the labels on an axis. The labels are the displayed values on the axis. Note: the title of an axis (returned by Axis.Title) is not an axis label.

Signature: C#

```
public abstract class AxisLabel : ChartText
```

Signature: VB.NET

```
Public MustInherit Class AxisLabel
  Inherits ChartText
```

Properties

Name	Description
Position	Sets or returns how the specified axis labels should be positioned in the chart.

Nested Classes

Name	Description
LabelPosition	A LabelPosition value specifies the position of an axis label in a chart.

AxisLabel.LabelPosition

Description

A LabelPosition value specifies the position of an axis label in a chart.

Signature: C#

```
public sealed class LabelPosition : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class LabelPosition
  Inherits TypesafeEnum
```

Values

Name	Description
HighEndPlotArea	The axis label will be displayed at the top of the chart's plot area.
Invisible	The axis label will not be shown.
LowEndPlotArea	The axis label will be displayed at the bottom of the chart's plot area.
NextToAxis	The axis label will be displayed next to the axis.

AxisLabel.Position

Description

Sets or returns how the specified axis labels should be positioned in the chart.

Signature: C#

```
public LabelPosition Position{ get; set; }
```

Signature: VB.NET

```
Public Property Position() As LabelPosition
```

AxisType

Description

When creating a chart with two X or Y axes (primary and secondary), use AxisType to associate a series with a particular axis.

Signature: C#

```
public sealed class AxisType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class AxisType
  Inherits TypesafeEnum
```

Values

Name	Description
Primary	Primary axis
Secondary	Secondary axis

Border

Description

A Border object represents a border around a cell or area of cells.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Border
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Border
```

Remarks

To set a border around a single cell, set the Border.Color and Border.Style of a `Border` object retrieved through the cell's Style:

```
cellB2.Style.Border.Style = Border.LineStyle.Thick;
```

```
cellB2.Style.Border.Color = Palette.SystemColor.Blue;
```

You can also define part of a cell's border using Border.Item(Border.Part):

```
cellB2.Style.Border[Border.Part.Outline].Color = Palette.SystemColor.Blue;
```

To return a `Border` object for an Area, use Area.BorderAround.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb[0];
          Area a = ws.CreateArea(4, 4, 15, 6);
          Border b = a.BorderAround;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb(0)
          Dim a As Area = ws.CreateArea(4, 4, 15, 6)
          Dim b As Border = a.BorderAround
```

Properties

Name	Description
Color	Sets or returns the color of the border, as an ExcelWriter `Color` object. If multiple colors were assigned to border parts, `Color` will return the color of the top border.
Style	Sets or returns the border's line style.

Indexers

Name	Description
Item(Border.Part)	Returns a BorderPart object representing the specified part of the border. This is the indexer for the Border object.

Nested Classes

Name	Description
LineStyle	Use the LineStyle class to set a line style for a Border or BorderPart.
Part	Use the Part class to set a color or line style for a specific part of a border.

Border.Color

Description

Sets or returns the color of the border, as an ExcelWriter `Color` object. If multiple colors were assigned to border parts, `Color` will return the color of the top border.

Signature: C#

```
public Color Color{ get; set; }
```

Signature: VB.NET

```
Public Property Color() As Color
```

Examples

C#

```
//--- Get Color
          Color c = b.Color;

          //--- Set Color
          b.Color = wb.Palette.GetClosestColor(System.Drawing.Color.Blue);
```

vb.net

```
'--- Get Color
          Dim c As Color = b.Color

          '--- Set Color
          b.Color = wb.Palette.GetClosestColor(System.Drawing.Color.Blue)
```

Border.Item(Border.Part)

Description

Returns a BorderPart object representing the specified part of the border. This is the indexer for the Border object.

Signature: C#

```
public BorderPart this[Part part] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal part As Part) As BorderPart
```

Parameters

part

The part of the border for which to set the line style.

Returns

The BorderPart object representing the specified part of the border.

Examples

C#

```
BorderPart bp = b[Border.Part.Left];
```

vb.net

```
Dim bp As BorderPart = b.Item(Border.Part.Left)
```

Border.LineStyle

Description

Use the LineStyle class to set a line style for a Border or BorderPart.

Signature: C#

```
public sealed class LineStyle : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class LineStyle
  Inherits TypesafeEnum
```

Values

Name	Description
Dashed	Dashed line
DashDot	Dash-dot line
DashDotDot	Dash-dot-dot line
Dotted	Dotted line
Double	Double line
Hair	Very thin line
Medium	Medium line
MediumDashed	Medium dashed line
MediumDashDot	Medium dash-dot line
MediumDashDotDot	Medium dash-dot-dot line
None	No line
SlantedDashDot	Slanted-dash-dot line
Thick	Thick line
Thin	Thin line

Border.Part

Description

Use the Part class to set a color or line style for a specific part of a border.

Signature: C#

```
public sealed class Part : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Part
  Inherits TypesafeEnum
```

Values

Name	Description

All	All borders and both diagonals
Bottom	Bottom border
DiagonalBoth	Both diagonal lines
DiagonalDown	Line from the top-left to the bottom-right
DiagonalUp	Line from the bottom-left to the top-right
Left	Left border
Outline	Left, right, top, and bottom borders
Right	Right border
Top	Top border

Border.Style

Description

Sets or returns the border's line style.

Signature: C#

```
public LineStyle Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As LineStyle
```

Remarks

To set the line style for a specific part of the border get the BorderPart object for the desired part of the border.

Examples

C#

```
//--- Get LineStyle
          Border.LineStyle ls = b.LineStyle;

          //--- Set LineStyle
          b.LineStyle = Border.LineStyle.Thick;
```

vb.net

```
'--- Get LineStyle
          Dim ls As Border.LineStyle = b.LineStyle

          '--- Set LineStyle
          b.LineStyle = Border.LineStyle.Thick
```

BorderPart

Description

A BorderPart object represents a particular part of a border around a cell or area of cells.

Signature: C#

```
public class BorderPart
```

Signature: VB.NET

```
Public Class BorderPart
```

Remarks

To get a `BorderPart`, use Border.Item.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Area a = ws.CreateArea(4, 4, 15, 6);
          Border b = a.BorderAround;
          BorderPart bp = b[Border.Part.Left];
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim a As Area = ws.CreateArea(4, 4, 15, 6)
          Dim b As Border = a.BorderAround
          Dim bp As BorderPart = b.Item(Border.Part.Left)
```

Properties

Name	Description
Color	Sets or returns the color of the border, as an ExcelWriter `Color` object. If multiple colors were assigned to border parts, `Color` will return the color of the top border.
Style	Sets or returns the border's line style.

BorderPart.Color

Description

Sets or returns the color of the border, as an ExcelWriter `Color` object. If multiple colors were assigned to border parts, `Color` will return the color of the top border.

Signature: C#

```
public Color Color{ get; set; }
```

Signature: VB.NET

```
Public Property Color() As Color
```

Examples

C#

```
//--- Get Color
          Color c = bp.Color;

          //--- Set Color
          bp.Color = wb.Palette.GetClosestColor(System.Drawing.Color.Blue);
```

vb.net

```
'--- Get Color
          Dim c As Color = bp.Color

          '--- Set Color
          bp.Color = wb.Palette.GetClosestColor(System.Drawing.Color.Blue)
```

BorderPart.Style

Description

Sets or returns the border's line style.

Signature: C#

```
public LineStyle Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As LineStyle
```

Examples

C#

```
//--- Get LineStyle
          Border.LineStyle ls = bp.Style;

          //---- Set LineStyle
          bp.Style = Border.LineStyle.Thin;
```

vb.net

```
'--- Get LineStyle
          Dim ls As Border.LineStyle = bp.Style

          '--- Set LineStyle
          bp.Style = Border.LineStyle.Thin
```

CategoryAxis

Description

A CategoryAxis object represents a chart's category (x) axis. `CategoryAxis` is a sub-class of Axis.

Signature: C#

```
public sealed class CategoryAxis : Axis
```

Signature: VB.NET

```
Public NotInheritable Class CategoryAxis
  Inherits Axis
```

Properties

Name	Description
AxisCrossesMidCategory	Sets or returns whether category axis major gridlines will be between categories, or in the middle of categories. Gridlines extend from the tick marks on an axis across the chart's plot area.
AxisCrossesMaxCategory	Sets or returns whether the value (y) axis crosses the category (x) axis at the maximum category value.
BaseUnit	Sets of returns time unit (day, month or year) setting used in to establish BaseUnit number
CategoriesBetweenTickMarks	Sets or returns the number of categories to show between axis tick marks.
CategoriesBetweenTickMarkLabels	Sets or returns the number of categories to show between tick mark labels.
DisplayType	Sets or returns the display type for the data on the category axis.
MajorUnit	Sets of returns time unit (day, month or year) setting used in to establish MajorUnitValue number
MajorUnitValue	Sets or returns a number, which represent Major Unit value
MaximumValue	Sets or returns the maximum value to be used in the category (x) axis
MaximumValueAsDate	Sets or returns the maximum value to be used in the category (x) axis, represented as DateTime
MinimumValue	Sets or returns the minimum value to be used in the category (x) axis
MinimumValueAsDate	Sets or returns the minimum value to be used in the category (x) axis, represented as DateTime
MinorUnit	Sets of returns time unit (day, month or year) setting used in to establish MinorUnitValue number
MinorUnitValue	Sets or returns a number, which represent Minor Unit value
ValueCrossingPoint	Sets or returns the category value at which the value (y) axis crosses the category (x) axis.
ValueCrossingPointAsDate	Sets or returns the category value at which the value (y) axis crosses the category (x) axis, represented as DateTime

Nested Classes

Name	Description
AxisDisplayType	Enumeration value allowing you to determine what type of values are displayed on this axis.
AxisTimeUnit	Enumeration for determinine what time unit to use for a chart axis

CategoryAxis.AxisCrossesMaxCategory

Description

Sets or returns whether the value (y) axis crosses the category (x) axis at the maximum category value.

Signature: C#

```
public boolean AxisCrossesMaxCategory{ get; set; }
```

Signature: VB.NET

```
Public Property AxisCrossesMaxCategory() As Boolean
```

CategoryAxis.AxisCrossesMidCategory

Description

Sets or returns whether category axis major gridlines will be between categories, or in the middle of categories. Gridlines extend from the tick marks on an axis across the chart's plot area.

Signature: C#

```
public boolean AxisCrossesMidCategory{ get; set; }
```

Signature: VB.NET

```
Public Property AxisCrossesMidCategory() As Boolean
```

CategoryAxis.AxisDisplayType

Description

Enumeration value allowing you to determine what type of values are displayed on this axis.

Signature: C#

```
public sealed class AxisDisplayType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class AxisDisplayType
  Inherits TypesafeEnum
```

Values

Name	Description
Automatic	Excel will set the display format automatically.
Category	Category axis values will be displayed as numbers.
TimeScale	Category axis values will be displayed as dates. A time-scale category axis displays dates in chronological order at specific intervals.

CategoryAxis.AxisTimeUnit

Description

Enumeration for determinine what time unit to use for a chart axis

Signature: C#

```
public sealed class AxisTimeUnit : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class AxisTimeUnit
  Inherits TypesafeEnum
```

Values

Name	Description
Day	Time is measured in days
Month	Time is measured in months

Year	Time is measured in years

CategoryAxis.BaseUnit

Description

Sets of returns time unit (day, month or year) setting used in to establish BaseUnit number

Signature: C#

```
public AxisTimeUnit BaseUnit{ get; set; }
```

Signature: VB.NET

```
Public Property BaseUnit() As AxisTimeUnit
```

CategoryAxis.CategoriesBetweenTickMarkLabels

Description

Sets or returns the number of categories to show between tick mark labels.

Signature: C#

```
public int CategoriesBetweenTickMarkLabels{ get; set; }
```

Signature: VB.NET

```
Public Property CategoriesBetweenTickMarkLabels() As Integer
```

CategoryAxis.CategoriesBetweenTickMarks

Description

Sets or returns the number of categories to show between axis tick marks.

Signature: C#

```
public int CategoriesBetweenTickMarks{ get; set; }
```

Signature: VB.NET

```
Public Property CategoriesBetweenTickMarks() As Integer
```

CategoryAxis.DisplayType

Description

Sets or returns the display type for the data on the category axis.

Signature: C#

```
public AxisDisplayType DisplayType{ get; set; }
```

Signature: VB.NET

```
Public Property DisplayType() As AxisDisplayType
```

CategoryAxis.MajorUnit

Description

Sets of returns time unit (day, month or year) setting used in to establish MajorUnitValue number

Signature: C#

```
public AxisTimeUnit MajorUnit{ get; set; }
```

Signature: VB.NET

```
Public Property MajorUnit() As AxisTimeUnit
```

CategoryAxis.MajorUnitValue

Description

Sets or returns a number, which represent Major Unit value

Signature: C#

```
public int MajorUnitValue{ get; set; }
```

Signature: VB.NET

```
Public Property MajorUnitValue() As Integer
```

CategoryAxis.MaximumValue

Description

Sets or returns the maximum value to be used in the category (x) axis

Signature: C#

```
public int MaximumValue{ get; set; }
```

Signature: VB.NET

```
Public Property MaximumValue() As Integer
```

CategoryAxis.MaximumValueAsDate

Description

Sets or returns the maximum value to be used in the category (x) axis, represented as DateTime

Signature: C#

```
public System.DateTime MaximumValueAsDate{ get; set; }
```

Signature: VB.NET

```
Public Property MaximumValueAsDate() As Date
```

CategoryAxis.MinimumValue

Description

Sets or returns the minimum value to be used in the category (x) axis

Signature: C#

```
public int MinimumValue{ get; set; }
```

Signature: VB.NET

```
Public Property MinimumValue() As Integer
```

CategoryAxis.MinimumValueAsDate

Description

Sets or returns the minimum value to be used in the category (x) axis, represented as DateTime

Signature: C#

```
public System.DateTime MinimumValueAsDate{ get; set; }
```

Signature: VB.NET

```
Public Property MinimumValueAsDate() As Date
```

CategoryAxis.MinorUnit

Description

Sets of returns time unit (day, month or year) setting used in to establish MinorUnitValue number

Signature: C#

```
public AxisTimeUnit MinorUnit{ get; set; }
```

Signature: VB.NET

```
Public Property MinorUnit() As AxisTimeUnit
```

CategoryAxis.MinorUnitValue

Description

Sets or returns a number, which represent Minor Unit value

Signature: C#

```
public int MinorUnitValue{ get; set; }
```

Signature: VB.NET

```
Public Property MinorUnitValue() As Integer
```

CategoryAxis.ValueCrossingPoint

Description

Sets or returns the category value at which the value (y) axis crosses the category (x) axis.

Signature: C#

```
public int ValueCrossingPoint{ get; set; }
```

Signature: VB.NET

```
Public Property ValueCrossingPoint() As Integer
```

CategoryAxis.ValueCrossingPointAsDate

Description

Sets or returns the category value at which the value (y) axis crosses the category (x) axis, represented as DateTime

Signature: C#

```
public System.DateTime ValueCrossingPointAsDate{ get; set; }
```

Signature: VB.NET

```
Public Property ValueCrossingPointAsDate() As Date
```

Cell

Description

A Cell object represents a single cell in a worksheet. To return a `Cell` object use <dotnet>Worksheet.Cells. Specify the cell by 0-based row and column indexes or by Excel-style reference:

Signature: C#

```
public sealed class Cell
```

Signature: VB.NET

```
Public NotInheritable Class Cell
```

Examples

C#

```
//--- Get Cell by Excel-style reference
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Cell cellA1 = ws.Cells[0,0];

          //--- Get Cell by Excel-style reference
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Cell cellA1 = ws.Cells["A1"];
```

vb.net

```
'--- Get Cell by Excel-style reference
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim cellA1 As Cell = ws.Cells(0, 0)

          '--- Get Cell by Excel-style reference
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim cellA1 As Cell = ws.Cells("A1")
```

Properties

Name	Description
ColumnNumber	Returns the cell's 0-based column number.
Comment	Returns a Comment object representing a note attached to the cell, separate from other cell content. If a comment is not associated with the specified cell, a new comment will be created and returned.
ConditionalFormat	Returns the ConditionalFormat object associated with the current Cell. If a ConditionalFormat object does not exist, a new one will be created. Any changes made to the object returned will only affect this `Cell`.
DataValidation	Returns the cell's DataValidation object if one exists. Otherwise, returns 'null'. When setting the dataValidation, if the value you're setting it to uses a local reference or area in a formula (i.e. "=A5:B7"), then the cell will adjust those local references accordingly based on the cell's location. The original dataValidation object you assigned the cell to use will remain unaffected.
Formula	Sets or returns a cell formula. If no formula is associated with the cell, the property returns an empty string.
HasComment	Returns `true` if a Comment object is associated with the cell.
HasConditionalFormat	Returns whether or not the Cell contains a ConditionalFormat object.
HasDataValidation	Returns `true` if the cell has a DataValidation object associated with it.
IsMerged	Returns whether or not the Cell is a merged cell.
Name	Returns the cell's Excel-style name (such as, "B5").
RowNumber	Returns the cell's 0-based row number.
Style	Sets or returns a Style object that determines how the cell looks, including number format, font, orientation, etc. Accessing the cell's style through Cell.Style will not affect the formatting of other cells with the same global style.
Value	Sets or returns a cell value.
ValueType	Returns the data type of the cell's value: "Blank, "Boolean", "Date", "Error", "Numeric", or "Text".

Methods

Name	Description
ApplyStyle(Style)	Applies a style to the cell. When a style is applied - rather than set (see Cell.Style) - only the differences between the new style and style properties previously assigned to the cell (through the ExcelWriter API) will take effect. For example, if the cell has a background color and the new style applied does not contain a background color, the cell's color will not be affected. However, if the new style includes a background color, it will replace the existing background color of the cell.
ClearContent()	Clears a cell's content. Calling this method is equivalent to setting a cell's value to null.
ClearRichTextFormatting()	Clears all rich text formatting within the cell. The font of the text will be set to that set in Cell.Style.Font.
CreateAnchor(Int32, Int32)	Creates an anchor within the specified cell. An anchor represents the position of a floating (non-cell data) object within a spreadsheet.

CreateHyperlink(String)	Creates a new hyperlink in the cell pointing to the specified href. This hyperlink is automatically added to the worksheet. The cell value will be converted to hyperlinks.
GetCharacters(Int32)	Returns a CharacterRun object representing a range of characters within the cell. Use this property to set the font on specified characters within the cell.
GetCharacters(Int32, Int32)	Returns a CharacterRun object representing a range of characters within the cell. Use this method to set the font on specified characters within the cell.
RemoveComment()	Removes the comment associated with the cell. A comment is a note attached to the cell, separate from other cell content. To find out if a comment is attached to the cell, use Cell.HasComment .
RemoveConditionalFormat()	Removes any ConditionalFormat objects from the current Cell.
RemoveDataValidation()	Removes the DataValidation object associated with the cell.
SetConditionalFormat(ConditionalFormat)	Applies the specified ConditionalFormat to the current Cell. If the cell already has a ConditionalFormat, it will be replaced by the specified `ConditionalFormat`.
ToString()	Returns a string representation of the Cell object.
Unmerge()	Unmerges the current cell. A merged cell is referenced using the location of the leftmost, topmost constituent cell.

Nested Classes

Name	Description
CellValueType	A CellValueType value specifies the data type of the cell's value.

Cell.ApplyStyle(Style)

Description

Applies a style to the cell. When a style is applied - rather than set (see Cell.Style) - only the differences between the new style and style properties previously assigned to the cell (through the ExcelWriter API) will take effect. For example, if the cell has a background color and the new style applied does not contain a background color, the cell's color will not be affected. However, if the new style includes a background color, it will replace the existing background color of the cell.

Signature: C#

```
public void ApplyStyle(Style style)
```

Signature: VB.NET

```
Public Sub ApplyStyle(ByVal style As Style)
```

Parameters

style

A Style object representing the style to apply to the cell.

Examples

C#

```
cellA1.ApplyStyle(s);
```

vb.net

```
cellA1.ApplyStyle(s)
```

Cell.CellValueType

Description

A CellValueType value specifies the data type of the cell's value.

Signature: C#

```
public sealed class CellValueType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class CellValueType
  Inherits TypesafeEnum
```

Values

Name	Description
Blank	The cell is empty.
Boolean	The cell contains a boolean value.
Currency	The cell contains a currency value.
Date	The cell value is a date.
Double	The cell value is a double.
Error	The cell contains an error value. Error values include #DIV/0!, #N/A, #NAME?, #NULL!, #NUM!, #REF!, and #VALUE!.
Integer	The cell value is an integer.
Text	The cell contains a string.
Time	The cell value is a time.

Cell.ClearContent()

Description

Clears a cell's content. Calling this method is equivalent to setting a cell's value to null.

Signature: C#

```
public void ClearContent()
```

Signature: VB.NET

```
Public Sub ClearContent()
```

Examples

C#

```
cellA1.ClearContent();
```

vb.net

```
cellA1.ClearContent()
```

Cell.ClearRichTextFormatting()

Description

Clears all rich text formatting within the cell. The font of the text will be set to that set in Cell.Style.Font.

Signature: C#

```
public void ClearRichTextFormatting()
```

Signature: VB.NET

```
Public Sub ClearRichTextFormatting()
```

Examples

C#

```
cellA1.ClearRichTextFormatting();
```

vb.net

```
cellA1.ClearRichTextFormatting()
```

Cell.ColumnNumber

Description

Returns the cell's 0-based column number.

Signature: C#

```
public int ColumnNumber{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ColumnNumber() As Integer
```

Examples

C#

```
int col = oCell.ColumnNumber;
```

vb.net

```
Dim col As Integer = oCell.ColumnNumber
```

Cell.Comment

Description

Returns a Comment object representing a note attached to the cell, separate from other cell content. If a comment is not associated with the specified cell, a new comment will be created and returned.

Signature: C#

```
public Comment Comment{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Comment() As Comment
```

Examples

C#

```
Comment cmt = cellA1.Comment;
```

vb.net

```
Dim cmt As Comment = cellA1.Comment
```

Cell.ConditionalFormat

Description

Returns the ConditionalFormat object associated with the current Cell. If a ConditionalFormat object does not exist, a new one will be created. Any changes made to the object returned will only affect this `Cell`.

Signature: C#

```
public ConditionalFormat ConditionalFormat{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ConditionalFormat() As ConditionalFormat
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Cell c = wb.Worksheets[0].Cells[0,0];
          ConditionalFormat cf = c.ConditionalFormat;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim c As Cell = wb.Worksheets(0).Cells(0,0)
          Dim cf As ConditionalFormat = c.ConditionalFormat
```

Cell.CreateAnchor(Int32, Int32)

Description

Creates an anchor within the specified cell. An anchor represents the position of a floating (non-cell data) object within a spreadsheet.

Signature: C#

```
public Anchor CreateAnchor(int percentX, int percentY)
```

Signature: VB.NET

```
Public Function CreateAnchor(ByVal percentX As Integer, ByVal percentY As Integer) As Anchor
```

Parameters

percentX

The X offset from the cell's left side as a percentage of the cell's width. This value must be between 0 and 100.

percentY

The Y offset from the top of the cell as a percentage of the cell's height. This value must be between 0 and 100.

Returns

An anchor for this cell.

Examples

C#

```
Anchor anch = cellB7.CreateAnchor(30, 50);
```

vb.net

```
Dim anch As Anchor = cellB7.CreateAnchor(30, 50)
```

Cell.CreateHyperlink(String)

Description

Creates a new hyperlink in the cell pointing to the specified href. This hyperlink is automatically added to the worksheet. The cell value will be converted to hyperlinks.

Signature: C#

```
public Hyperlink CreateHyperlink(System.String href)
```

Signature: VB.NET

```
Public Function CreateHyperlink(ByVal href As String) As Hyperlink
```

Parameters

href

The hyperlink's destination.

Returns

A Hyperlink object.

Examples

C#

```
Hyperlink link = cellA1.CreateHyperlink("http://www.softartisans.com");
```

vb.net

```
Dim link As Hyperlink = cellA1.CreateHyperlink("http://www.softartisans.com")
```

Cell.DataValidation

Description

Returns the cell's DataValidation object if one exists. Otherwise, returns 'null'.

When setting the dataValidation, if the value you're setting it to uses a local reference or area in a formula (i.e. "=A5:B7"), then the cell will adjust those local references accordingly based on the cell's location. The original dataValidation object you assigned the cell to use will remain unaffected.

Signature: C#

```
public DataValidation DataValidation{ get; set; }
```

Signature: VB.NET

```
Public Property DataValidation() As DataValidation
```

Examples

C#

```
DataValidation dv = cellA1.DataValidation;
```

vb.net

```
Dim dv As DataValidation = cellA1.DataValidation
```

Cell.Formula

Description

Sets or returns a cell formula. If no formula is associated with the cell, the property returns an empty string.

Signature: C#

```
public System.String Formula{ get; set; }
```

Signature: VB.NET

```
Public Property Formula() As String
```

Examples

C#

```
//--- Get CellFormula
          string cf = cellA1.Formula;

          //--- Set CellFormula
          cellA1.Formula = "=SUM(A1:A2)";
```

vb.net

```
'--- Get CellFormula
          Dim cf As string = cellA1.Formula

          '--- Set CellFormula
          oCell.Formula = "=SUM(A1:A2)"
```

Cell.GetCharacters

Overload	Description
GetCharacters(Int32)	Returns a CharacterRun object representing a range of characters within the cell. Use this property to set the font on specified characters within the cell.
GetCharacters(Int32, Int32)	Returns a CharacterRun object representing a range of characters within the cell. Use this method to set the font on specified characters within the cell.

Cell.GetCharacters(Int32, Int32)

Description

Returns a CharacterRun object representing a range of characters within the cell. Use this method to set the font on specified characters within the cell.

Signature: C#

```
public CharacterRun GetCharacters(int startIndex, int length)
```

Signature: VB.NET

```
Public Function GetCharacters(ByVal startIndex As Integer, ByVal length As Integer) As CharacterRun
```

Parameters

startIndex

The character position at which to begin the character run.

length

The length of the character run.

Returns

A `CharacterRun` object representing a range of characters within the cell.

Examples

C#

```
CharacterRun charRun = cellA1.GetCharacters(0, 6);
```

vb.net

```
Dim charRun As CharacterRun = cellA1.GetCharacters(0, 6)
```

Cell.GetCharacters(Int32)

Description

Returns a CharacterRun object representing a range of characters within the cell. Use this property to set the font on specified characters within the cell.

Signature: C#

```
public CharacterRun GetCharacters(int startIndex)
```

Signature: VB.NET

```
Public Function GetCharacters(ByVal startIndex As Integer) As CharacterRun
```

Parameters

startIndex

The character position at which to begin the character run.

Returns

A `CharacterRun` object representing a range of characters within the cell.

Examples

C#

```
CharacterRun charRun = cellA1.GetCharacters(0);
```

vb.net

```
Dim charRun As CharacterRun = cellA1.GetCharacters(0)
```

Cell.HasComment

Description

Returns `true` if a Comment object is associated with the cell.

Signature: C#

```
public boolean HasComment{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property HasComment() As Boolean
```

Examples

C#

```
bool noteAttached = cellA1.HasComment;
```

vb.net

```
Dim noteAttached As Boolean = cellA1.HasComment
```

Cell.HasConditionalFormat

Description

Returns whether or not the Cell contains a ConditionalFormat object.

Signature: C#

```
public boolean HasConditionalFormat{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property HasConditionalFormat() As Boolean
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          bool hasCndFmt = wb.Worksheets[0].Cells[0,0].HasConditionalFormat;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          bool hasCndFmt = wb.Worksheets(0).Cells(0,0).HasConditionalFormat
```

Cell.HasDataValidation

Description

Returns `true` if the cell has a DataValidation object associated with it.

Signature: C#

```
public boolean HasDataValidation{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property HasDataValidation() As Boolean
```

Examples

C#

```
bool cellA1HasDV = cellA1.HasDataValidation;
```

vb.net

```
Dim cellA1HasDV As Boolean = cellA1.HasDataValidation
```

Cell.IsMerged

Description

Returns whether or not the Cell is a merged cell.

Signature: C#

```
public boolean IsMerged{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property IsMerged() As Boolean
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          bool isMerged = wb.Worksheets[0].Cells[0, 0].IsMerged;
```

vb.net

```
Dim xla As New ExcelApplication()
          bool isMerged = wb.Worksheets(0).Cells(0, 0).IsMerged
```

Cell.Name

Description

Returns the cell's Excel-style name (such as, "B5").

Signature: C#

```
public System.String Name{ get; }
```

Signature: VB.NET

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

Examples

C#

```
string cellName = cellA1.Name;
```

vb.net

```
Dim cellName As String = cellA1.Name
```

Cell.RemoveComment()

Description

Removes the comment associated with the cell. A comment is a note attached to the cell, separate from other cell content. To find out if a comment is attached to the cell, use Cell.HasComment .

Signature: C#

```
public void RemoveComment()
```

Signature: VB.NET

```
Public Sub RemoveComment()
```

Examples

C#

```
cellA1.RemoveComment();
```

vb.net

```
cellA1.RemoveComment()
```

Cell.RemoveConditionalFormat()

Description

Removes any ConditionalFormat objects from the current Cell.

Signature: C#

```
public void RemoveConditionalFormat()
```

Signature: VB.NET

```
Public Sub RemoveConditionalFormat()
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          wb.Worksheets[0].Cells[0,0].RemoveConditionalFormat();
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          wb.Worksheets(0).Cells(0,0).RemoveConditionalFormat()
```

Cell.RemoveDataValidation()

Description

Removes the DataValidation object associated with the cell.

Signature: C#

```
public void RemoveDataValidation()
```

Signature: VB.NET

```
Public Sub RemoveDataValidation()
```

Examples

C#

```
cell.RemoveDataValidation();
```

vb.net

```
cell.RemoveDataValidation()
```

Cell.RowNumber

Description

Returns the cell's 0-based row number.

Signature: C#

```
public int RowNumber{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property RowNumber() As Integer
```

Examples

C#

```
int row = cellA1.RowIndex;
```

vb.net

```
Dim row As Integer = cellA1.RowIndex
```

Cell.SetAddInFormula(String)

Description

Sets the formula of the worksheet to the given Add-In Formula.

Signature: C#

```
public void SetAddInFormula(string formula)
```

Signature: VB.NET

```
Public Sub SetAddInFormula(ByVal formula As String)
```

Parameters

formula

Add-In Formula you wish to add, as a String. Include the equals sign as well as a reference to the Excel Add-In file you wish to reference, if the file is not included in the local macros directory.

Remarks

Examples

C#

```
//--- Set Cell D3 on the first Worksheet to reference formula Module1.Apple
//--- from the file TestAddInFormula2Helper.xpl.xla
WB[0]["D3"].SetAddInFormula("=TestAddInFormula2Helper.xpl.xla!Module1.Apple(1)");
```

vb.net

```
'--- Set Cell D3 on the first Worksheet to reference formula Module1.Apple
'--- from the file TestAddInFormula2Helper.xpl.xla
WB(0)("D3").SetAddInFormula("=TestAddInFormula2Helper.xpl.xla!Module1.Apple(1)")
```

Cell.SetConditionalFormat(ConditionalFormat)

Description

Applies the specified ConditionalFormat to the current Cell. If the cell already has a ConditionalFormat, it will be replaced by the specified `ConditionalFormat`.

Signature: C#

```
public void SetConditionalFormat(ConditionalFormat conditionalFormat)
```

Signature: VB.NET

```
Public Sub SetConditionalFormat(ByVal conditionalFormat As ConditionalFormat)
```

Parameters

conditionalFormat

The `ConditionalFormat` to apply to the cell.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond = condFmt.CreateCondition(Condition.Comparison.CellValueGreaterThan, "=100");
          Font fnt = cond.Style.Font;
          fnt.Bold = true;
          fnt.Color = wb.Palette.GetClosestColor(100,100,255);
          ws.Cells[0,0].SetConditionalFormat(condFmt);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim  condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = condFmt.CreateCondition(Condition.Comparison.CellValueGreaterThan,
"=100")
          Dim fnt As Font = cond.Style.Font
          fnt.Bold = True
          fnt.Color = wb.Palette.GetClosestColor(100,100,255)
          ws.Cells(0,0).SetConditionalFormat(condFmt)
```

Cell.Style

Description

Sets or returns a Style object that determines how the cell looks, including number format, font, orientation, etc. Accessing the cell's style through Cell.Style will not affect the formatting of other cells with the same global style.

Signature: C#

```
public Style Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As Style
```

Remarks

When a style is set - rather than applied (see Cell.ApplyStyle) - all properties of the style will be assigned to the cell. Properties not explicitly set on the new style default to Normal style values.

Examples

C#

```
//--- Get Style
          Style s = cellA1.Style;

          //--- Set Style
          cellA1.Style = s;
```

vb.net

```
'--- Get Style
          Dim s As Style = cellA1.Style

          '--- Set Style
          cellA1.Style = s
```

Cell.ToString()

Description

Returns a string representation of the Cell object.

Signature: C#

```
public virtual System.String ToString()
```

Signature: VB.NET

```
Public Overridable Function ToString() As String
```

Returns

A string representation of the `Cell` object.

Examples

C#

```
string color = cell.ToString();
```

vb.net

```
Dim color As String = cell.ToString()
```

Cell.Unmerge()

Description

Unmerges the current cell. A merged cell is referenced using the location of the leftmost, topmost constituent cell.

Signature: C#

```
public void Unmerge()
```

Signature: VB.NET

```
Public Sub Unmerge()
```

Cell.Value

Description

Sets or returns a cell value.

Signature: C#

```
public System.Object Value{ get; set; }
```

Signature: VB.NET

```
Public Property Value() As Object
```

Examples

C#

```
//--- Get Value
          Object cellValue = cellA1.Value;

          //--- Set Value
          cellA1.Value = "value";
```

vb.net

```
'--- Get Value
          Dim cellValue As Object = cellA1.Value

          '--- Set Value
          cellA1.Value = "value"
```

Cell.ValueType

Description

Returns the data type of the cell's value: "Blank, "Boolean", "Date", "Error", "Numeric", or "Text".

Signature: C#

```
public CellValueType ValueType{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ValueType() As CellValueType
```

Examples

C#

```
Cell.CellValueType cvt = cellA1.ValueType;
```

vb.net

```
Dim cvt As Cell.CellValueType = cellA1.ValueType
```

Cells

Description

A Cells collection represents the set of cells in a worksheet.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Cells
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Cells
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Cells cls = ws.Cells;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim cls As Cells = ws.Cells
```

Indexers

Name	Description
Item(Int32, Int32)	Represents a single cell in a Cells collection. Item is the indexer of the `Cells` collection, so `Cells[0, 0]` is equivalent to `Cells.Item[0, 0]`.
Item(String)	Represents a single cell in a Cells collection. Item is the indexer of the `Cells` collection, so `Cells["A1"]` is equivalent to `Cells.Item["A1"]`.

Cells.Item

Overload	Description
Item(Int32, Int32)	Represents a single cell in a Cells collection. Item is the indexer of the `Cells` collection, so `Cells[0, 0]` is equivalent to `Cells.Item[0, 0]`.
Item(String)	Represents a single cell in a Cells collection. Item is the indexer of the `Cells` collection, so `Cells["A1"]` is equivalent to `Cells.Item["A1"]`.

Cells.Item(Int32, Int32)

Description

Represents a single cell in a Cells collection. Item is the indexer of the `Cells` collection, so `Cells[0, 0]` is equivalent to `Cells.Item[0, 0]`.

Signature: C#

```
public Cell this[int rowNumber, int columnNumber] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal rowNumber As Integer, ByVal columnNumber As Integer) As
Cell
```

Parameters

rowNumber

The 0-based row number of the cell to retrieve.

columnNumber

The 0-based column number of the cell to retrieve.

Returns

A Cell object representing the cell at the specified position.

Examples

C#

```
Cell cellA1 = cls[0, 0];
```

vb.net

```
Dim cellA1 As Cell = cls(0, 0)
```

Cells.Item(String)

Description

Represents a single cell in a Cells collection. Item is the indexer of the `Cells` collection, so `Cells["A1"]` is equivalent to `Cells.Item["A1"]`.

Signature: C#

```
public Cell this[System.String name] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal name As String) As Cell
```

Parameters

index

The index of the cell as a string

Returns

A Cell object representing the cell at the specified position.

Examples

C#

```
Cell cellA1 = cls["A1"];
```

vb.net

```
Dim cellA1 As Cell = cls("A1")
```

CellStyle

Description

The unique cell style for a given cell.

Signature: C#

```
public sealed class CellStyle : Style
```

Signature: VB.NET

```
Public NotInheritable Class CellStyle
  Inherits Style
```

Remarks

Every cell in an Excel workbook contains a unique cell style. This style is exposed by ExcelWriter through the CellStyle class. Changes to a `CellStyle` affect *only* the cell that owns the style. A `CellStyle` may be "set" on another cell or group of cells, but this action clones the style. Changes to the original `CellStyle` reference affect only the owning cell and changes to the cloned `CellStyle` affect only the cell on which the style was set. If the style is "set" on a cell grouping object, such as an area, range, row, or column, each cell in the grouping receives a unique clone of the style.

CharacterRun

Description

The CharacterRun class allows you to set a font on a specified range of characters within a ChartText object or a cell. To return a `CharacterRun` object, use Cell.GetCharacters() or ChartText.GetCharacters().

Signature: C#

```
public sealed class CharacterRun
```

Signature: VB.NET

```
Public NotInheritable Class CharacterRun
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Cell cellA1 = ws.Cells["A1"];
          cellA1.Value = "Test";
          CharacterRun charRun = cellA1.GetCharacters(0);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim cellA1 As Cell = ws.Cells("A1")
          cellA1.Value = "Test"
          Dim charRun As CharacterRun = cellA1.GetCharacters(0)
```

Properties

Name	Description
Font	Sets or returns the font for the CharacterRun.
Length	Returns the length - in characters - of the CharacterRun.
StartIndex	Returns the 0-based index of the first character in the CharacterRun.

Methods

Name	Description
Clear()	Sets the font of the CharacterRun to the main font of the cell or ChartText object.

CharacterRun.Clear()

Description

Sets the font of the CharacterRun to the main font of the cell or ChartText object.

Signature: C#

```
public void Clear()
```

Signature: VB.NET

```
Public Sub Clear()
```

Examples

C#

```
charRun.Clear();
```

vb.net

```
charRun.Clear()
```

CharacterRun.Font

Description

Sets or returns the 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.Length

Description

Returns the length - in characters - of the CharacterRun.

Signature: C#

```
public int Length{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Length() As Integer
```

Examples

C#

```
int numChars = oCharacterRun.Length;
```

vb.net

```
Dim numChars As Integer = oCharacterRun.Length
```

CharacterRun.StartIndex

Description

Returns the 0-based index of the first character in the CharacterRun.

Signature: C#

```
public int StartIndex{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property StartIndex() As Integer
```

Examples

C#

```
int firstChar = charRun.StartIndex;
```

vb.net

```
Dim firstChar As Integer = charRun.StartIndex
```

Chart

Description

The Chart class represents an Excel chart. It is the main class for code-based chart creation and modification.

Signature: C#

```
public sealed class Chart
```

Signature: VB.NET

```
Public NotInheritable Class Chart
```

Remarks

To create a `Chart` object:

Properties

Name	Description
CategoryAxis	Returns a CategoryAxis object representing the chart's X axis.
ChartArea	Returns a ChartRegion object, representing the chart area in the worksheet. The chart area is the entire area within the worksheet assigned to the chart.
ChartSettings3D	Returns a ChartSettings3D object representing the properties of a three-dimensional chart.
ChartType	Sets or returns the chart's type (Area, Column, Bubble, etc.).
DataTable	For all chart types except Pie, Doughnut, Bubble, and Surface, `DataTable` returns a ChartDataTable object, representing a chart's data table.
Height	Sets or returns the height of the chart, in points (1/72 inch).
Legend	Returns a Legend object, representing the chart's legend.
Name	Sets or returns the name of the chart.
ObjectPositioning	Sets or returns the ObjectPositioning value that determines the chart's resize and reposition behavior when the cells in the worksheet are changed.
PageSetup	Returns the ChartPageSetup object for this chart.
PlotArea	Returns a ChartRegion object, representing the chart's plot area, the area of the chart itself.
Position	Sets or returns the chart's position in the worksheet.

PositionInCollection	Returns the 0-based position in the parent Charts collection. If this Chart is on its own Chartsheet, then this will return 0.
ScatterValueAxis	Returns a ValueAxis object representing the chart's scatter values on the X axis.
SecondaryCategoryAxis	Returns a CategoryAxis object, representing a chart's secondary X axis.
SecondaryValueAxis	Returns a ValueAxis object, representing a chart's secondary Y axis.
SeriesCollection	Returns a SeriesCollection object representing the set of data series in the chart.
ShowRoundedCorners	Sets or returns whether the chart area will be sharp-cornered or round-cornered.
Title	Returns a ChartText object representing the chart's title.
ValueAxis	Returns a ValueAxis object, representing a chart's Y axis.
Width	Sets or returns the width of the chart, in points (1/72 inch).
ZCategoryAxis	Returns a CategoryAxis object representing the chart's category values on the Z axis.

Methods

Name	Description
MoveChart(Chart.ChartLocation, Int32)	Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.
MoveChart(Chart.ChartLocation, Int32, Anchor)	Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.
MoveChart(Chart.ChartLocation, String)	Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.
MoveChart(Chart.ChartLocation, String, Int32)	Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.

Nested Classes

Name	Description
ChartLocation	A ChartLocation value specifies whether a chart should be an object within a worksheet, or converted to a chart sheet.

Chart.CategoryAxis

Description

Returns a CategoryAxis object representing the chart's X axis.

Signature: C#

```
public CategoryAxis CategoryAxis{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property CategoryAxis() As CategoryAxis
```

Chart.ChartArea

Description

Returns a ChartRegion object, representing the chart area in the worksheet. The chart area is the entire area within the worksheet assigned to the chart.

Signature: C#

```
public ChartRegion ChartArea{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ChartArea() As ChartRegion
```

Chart.ChartLocation

Description

A ChartLocation value specifies whether a chart should be an object within a worksheet, or converted to a chart sheet.

Signature: C#

```
public sealed class ChartLocation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ChartLocation
  Inherits TypesafeEnum
```

Values

Name	Description
ChartAsSheet	The chart should be converted to a chart sheet.
ObjectInSheet	The chart should be an object in the specified worksheet.

Chart.ChartSettings3D

Description

Returns a ChartSettings3D object representing the properties of a three-dimensional chart.

Signature: C#

```
public ChartSettings3D ChartSettings3D{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ChartSettings3D() As ChartSettings3D
```

Chart.ChartType

Description

Sets or returns the chart's type (Area, Column, Bubble, etc.).

Signature: C#

```
public ChartType ChartType{ get; set; }
```

Signature: VB.NET

```
Public Property ChartType() As ChartType
```

Chart.DataTable

Description

For all chart types except Pie, Doughnut, Bubble, and Surface, `DataTable` returns a ChartDataTable object, representing a chart's data table.

Signature: C#

```
public ChartDataTable DataTable{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DataTable() As ChartDataTable
```

Chart.Height

Description

Sets or returns the height of the chart, in points (1/72 inch).

Signature: C#

```
public double Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Double
```

Chart.Legend

Description

Returns a Legend object, representing the chart's legend.

Signature: C#

```
public Legend Legend{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Legend() As Legend
```

Chart.MoveChart

Overload	Description
MoveChart(Chart.ChartLocation, Int32)	Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.
MoveChart(Chart.ChartLocation, Int32, Anchor)	Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.
MoveChart(Chart.ChartLocation, String)	Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.
MoveChart(Chart.ChartLocation, String, Int32)	Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.

Chart.MoveChart(Chart.ChartLocation, Int32, Anchor)

Description

Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.

Signature: C#

```
public void MoveChart(ChartLocation chartLocation, int sheetIndex, Anchor anchor)
```

Signature: VB.NET

```
Public Sub MoveChart(ByVal chartLocation As ChartLocation, ByVal sheetIndex As Integer, ByVal anchor
As Anchor)
```

Parameters

chartLocation

Chart.ChartLocation.ChartAsSheet or Chart.ChartLocation.ObjectInSheet.

sheetIndex

The 0-based index of the worksheet to move the chart to or the index of the chartsheet to insert.

anchor

The postion of the top left corner of the chart in the worksheet.

Chart.MoveChart(Chart.ChartLocation, Int32)

Description

Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.

Signature: C#

```
public void MoveChart(ChartLocation chartLocation, int sheetIndex)
```

Signature: VB.NET

```
Public Sub MoveChart(ByVal chartLocation As ChartLocation, ByVal sheetIndex As Integer)
```

Parameters

chartLocation

Chart.ChartLocation.ChartAsSheet or Chart.ChartLocation.ObjectInSheet.

sheetIndex

The 0-based index of the worksheet to move the chart to or the index of the chartsheet to insert.

Chart.MoveChart(Chart.ChartLocation, String, Int32)

Description

Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.

Signature: C#

```
public void MoveChart(ChartLocation chartLocation, System.String sheetName, int sheetIndex)
```

Signature: VB.NET

```
Public Sub MoveChart(ByVal chartLocation As ChartLocation, ByVal sheetName As String, ByVal sheetIndex
As Integer)
```

Parameters

chartLocation

Chart.ChartLocation.ChartAsSheet or Chart.ChartLocation.ObjectInSheet.

sheetName

New name for the sheet to which the chart is moved.

sheetIndex

The 0-based index of the worksheet to move the chart to or the index of the chartsheet to insert.

Chart.MoveChart(Chart.ChartLocation, String)

Description

Moves the chart to a different worksheet. The method specifies whether the chart should be inserted as an object within the worksheet, or converted to a chart sheet.

Signature: C#

```
public void MoveChart(ChartLocation chartLocation, System.String sheetName)
```

Signature: VB.NET

```
Public Sub MoveChart(ByVal chartLocation As ChartLocation, ByVal sheetName As String)
```

Parameters

chartLocation

Chart.ChartLocation.ChartAsSheet or Chart.ChartLocation.ObjectInSheet.

sheetName

New name for the sheet to which the chart is moved.

Chart.Name

Description

Sets or returns the name of the chart.

Signature: C#

```
public System.String Name{ get; set; }
```

Signature: VB.NET

```
Public Property Name() As String
```

Chart.ObjectPositioning

Description

Sets or returns the ObjectPositioning value that determines the chart's resize and reposition behavior when the cells in the worksheet are changed.

Signature: C#

```
public ObjectPositioning ObjectPositioning{ get; set; }
```

Signature: VB.NET

```
Public Property ObjectPositioning() As ObjectPositioning
```

Chart.PageSetup

Description

Returns the ChartPageSetup object for this chart.

Signature: C#

```
public ChartPageSetup PageSetup{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PageSetup() As ChartPageSetup
```

Chart.PlotArea

Description

Returns a ChartRegion object, representing the chart's plot area, the area of the chart itself.

Signature: C#

```
public ChartRegion PlotArea{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PlotArea() As ChartRegion
```

Chart.Position

Description

Sets or returns the chart's position in the worksheet.

Signature: C#

```
public Anchor Position{ get; set; }
```

Signature: VB.NET

```
Public Property Position() As Anchor
```

Chart.PositionInCollection

Description

Returns the 0-based position in the parent Charts collection. If this Chart is on its own Chartsheet, then this will return 0.

Signature: C#

```
public int PositionInCollection{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PositionInCollection() As Integer
```

Chart.ScatterValueAxis

Description

Returns a ValueAxis object representing the chart's scatter values on the X axis.

Signature: C#

```
public ValueAxis ScatterValueAxis{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ScatterValueAxis() As ValueAxis
```

Chart.SecondaryCategoryAxis

Description

Returns a CategoryAxis object, representing a chart's secondary X axis.

Signature: C#

```
public CategoryAxis SecondaryCategoryAxis{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property SecondaryCategoryAxis() As CategoryAxis
```

Chart.SecondaryValueAxis

Description

Returns a ValueAxis object, representing a chart's secondary Y axis.

Signature: C#

```
public ValueAxis SecondaryValueAxis{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property SecondaryValueAxis() As ValueAxis
```

Chart.SeriesCollection

Description

Returns a SeriesCollection object representing the set of data series in the chart.

Signature: C#

```
public SeriesCollection SeriesCollection{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property SeriesCollection() As SeriesCollection
```

Chart.ShowRoundedCorners

Description

Sets or returns whether the chart area will be sharp-cornered or round-cornered.

Signature: C#

```
public boolean ShowRoundedCorners{ get; set; }
```

Signature: VB.NET

```
Public Property ShowRoundedCorners() As Boolean
```

Chart.Title

Description

Returns a ChartText object representing the chart's title.

Signature: C#

```
public ChartText Title{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Title() As ChartText
```

Chart.ValueAxis

Description

Returns a ValueAxis object, representing a chart's Y axis.

Signature: C#

```
public ValueAxis ValueAxis{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ValueAxis() As ValueAxis
```

Chart.Width

Description

Sets or returns the width of the chart, in points (1/72 inch).

Signature: C#

```
public double Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Double
```

Chart.ZCategoryAxis

Description

Returns a CategoryAxis object representing the chart's category values on the Z axis.

Signature: C#

```
public CategoryAxis ZCategoryAxis{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ZCategoryAxis() As CategoryAxis
```

ChartDataTable

Description

A ChartDataTable object represents a chart's data table. A data table is a grid that can be added to some charts that contains the numeric data used to create the chart. The data table is usually attached to the chart's category axis, replacing the tick-mark labels on the axis.

Signature: C#

```
public sealed class ChartDataTable
```

Signature: VB.NET

```
Public NotInheritable Class ChartDataTable
```

Remarks

To return a `ChartDataTable`, use Chart.DataTable.

Properties

Name	Description
BorderLine	Returns a ChartLine object representing the border around the data table.
Font	Sets or returns the font for text in the data table.
Height	Sets or returns the height of the data table, in units of 1/4000 of the chart area.
ShowHorizontalBorders	Sets or returns whether the chart's data table will display horizontal grid lines.
ShowLegendKeys	Sets or returns whether the data table will include legend keys next to series titles.
ShowOutlineBorders	Sets or returns whether the data table will display outline gridlines.
ShowVerticalBorders	Sets or returns whether the chart's data table will display vertical grid lines.
Visible	Sets or returns whether to display a data table (in place of category axis tick mark labels). Data tables are not available for all chart types.

Width	Sets or returns the width of the data table, in units of 1/4000 of the chart area.
X	Sets or returns the x coordinate of the upper left corner of the data table, in units of 1/4000 of the chart area.
Y	Sets or returns the y coordinate of the upper left corner of the data table, in units of 1/4000 of the chart area.

ChartDataTable.BorderLine

Description

Returns a ChartLine object representing the border around the data table.

Signature: C#

```
public ChartLine BorderLine{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property BorderLine() As ChartLine
```

ChartDataTable.Font

Description

Sets or returns the font for text in the data table.

Signature: C#

```
public Font Font{ get; set; }
```

Signature: VB.NET

```
Public Property Font() As Font
```

ChartDataTable.Height

Description

Sets or returns the height of the data table, in units of 1/4000 of the chart area.

Signature: C#

```
public int Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Integer
```

ChartDataTable.ShowHorizontalBorders

Description

Sets or returns whether the chart's data table will display horizontal grid lines.

Signature: C#

```
public boolean ShowHorizontalBorders{ get; set; }
```

Signature: VB.NET

```
Public Property ShowHorizontalBorders() As Boolean
```

ChartDataTable.ShowLegendKeys

Description

Sets or returns whether the data table will include legend keys next to series titles.

Signature: C#

```
public boolean ShowLegendKeys{ get; set; }
```

Signature: VB.NET

```
Public Property ShowLegendKeys() As Boolean
```

ChartDataTable.ShowOutlineBorders

Description

Sets or returns whether the data table will display outline gridlines.

Signature: C#

```
public boolean ShowOutlineBorders{ get; set; }
```

Signature: VB.NET

```
Public Property ShowOutlineBorders() As Boolean
```

ChartDataTable.ShowVerticalBorders

Description

Sets or returns whether the chart's data table will display vertical grid lines.

Signature: C#

```
public boolean ShowVerticalBorders{ get; set; }
```

Signature: VB.NET

```
Public Property ShowVerticalBorders() As Boolean
```

ChartDataTable.Visible

Description

Sets or returns whether to display a data table (in place of category axis tick mark labels). Data tables are not available for all chart types.

Signature: C#

```
public boolean Visible{ get; set; }
```

Signature: VB.NET

```
Public Property Visible() As Boolean
```

ChartDataTable.Width

Description

Sets or returns the width of the data table, in units of 1/4000 of the chart area.

Signature: C#

```
public int Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Integer
```

ChartDataTable.X

Description

Sets or returns the x coordinate of the upper left corner of the data table, in units of 1/4000 of the chart area.

Signature: C#

```
public int X{ get; set; }
```

Signature: VB.NET

```
Public Property X() As Integer
```

Remarks

For SeriesDataLabel s on a column or bar chart X is the offset perpendicular to the bar or column from the default position in 1/4000 of the ChartArea. For `SeriesDataLabel` s on a pie chart X is offset angle from the default position in radians. For `SeriesDataLabel` s in all other charts types X is the offset from the default x position in 1/4000 of the ChartArea.

ChartDataTable.Y

Description

Sets or returns the y coordinate of the upper left corner of the data table, in units of 1/4000 of the chart area.

Signature: C#

```
public int Y{ get; set; }
```

Signature: VB.NET

```
Public Property Y() As Integer
```

ChartFillPattern

Description

Signature: C#

```
public class ChartFillPattern
```

Signature: VB.NET

```
Public Class ChartFillPattern
```

Values

Name	Description
None	No Pattern
Solid	Solid color
Hatch5Percent	5%
Hatch50Percent	50%
LightDownwardDiagonal	Light Downward Diagonal
LightVertical	Light Vertical
DashedDownwardDiagonal	Dashed Downward Diagonal
ZigZag	Zig zag
Divot	Divot
SmallGrid	Small grid
Hatch10Percent	10%
Hatch60Percent	60%
LightUpwardDiagonal	Light Upward Diagonal
LightHorizontal	Light Horizontal
DashedUpwardDiagonal	Dashed Upward Diagonal
Wave	Wave
DottedGrid	Dotted Grid
LargeGrid	Large Grid
Hatch20Percent	20%
Hatch70Percent	70%

DarkDownwardDiagonal	Dark Downward Diagonal
NarrowVertical	Narrow Vertical
DashedHorizontal	Dashed Horizontal
DiagonalBrick	Diagonal Brick
DottedDiamond	Dotted Diamond
SmallCheckerBoard	Small Checker Board
Hatch25Percent	25%
Hatch75Percent	75%
DarkUpwardDiagonal	Dark Upward Diagonal
NarrowHorizontal	Narrow Horizontal
DashedVertical	Dashed Vertical
HorizontalBrick	Horizontal Brick
Shingle	Shingle
LargeCheckerBoard	Large Checker Board
Hatch30Percent	30%
Hatch80Percent	80%
WideDownwardDiagonal	Wide downward diagonal
DarkVertical	Dark Vertical
SmallConfetti	Small Confetti
Weave	Weave
Trellis	Trellis
OutlinedDiamond	Outlined Diamond
Hatch40Percent	40%
Hatch90Percent	90%

WideUpwardDiagonal	Wide Upward Diagonal
DarkHorizontal	Dark Horizontal
LargeConfetti	Large Confetti
Plaid	Plaid
Sphere	Sphere
SolidDiamond	Solid Diamond

ChartInterior

Description

A ChartInterior object contains formatting properties of a ChartRegion or an object extending `ChartRegion` (AxisLabel, ChartText, Legend, or SeriesDataLabel).

Signature: C#

```
public sealed class ChartInterior
```

Signature: VB.NET

```
Public NotInheritable Class ChartInterior
```

Remarks

property of the following classes: `ChartRegion`, `AxisLabel`, `ChartText`, `Legend`, and `SeriesDataLabel`.

Properties

Name	Description
BackgroundColor	Sets or returns the background color of the specified chart region.
ForegroundColor	Sets or returns the foreground color of the specified chart region.
Pattern	Sets or returns the background pattern of the specified chart region.
Visible	Sets or returns whether the specified chart region is visible.

Methods

Name	Description
ClearFillEffects()	Clears any fill effect that may be present in the chart interior.

ChartInterior.BackgroundColor

Description

Sets or returns the background color of the specified chart region.

Signature: C#

```
public Color BackgroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property BackgroundColor() As Color
```

ChartInterior.ClearFillEffects()

Description

Clears any fill effect that may be present in the chart interior.

Signature: C#

```
public void ClearFillEffects()
```

Signature: VB.NET

```
Public Sub ClearFillEffects()
```

ChartInterior.ForegroundColor

Description

Sets or returns the foreground color of the specified chart region.

Signature: C#

```
public Color ForegroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property ForegroundColor() As Color
```

ChartInterior.Pattern

Description

Sets or returns the background pattern of the specified chart region.

Signature: C#

```
public Pattern Pattern{ get; set; }
```

Signature: VB.NET

```
Public Property Pattern() As Pattern
```

ChartInterior.Visible

Description

Sets or returns whether the specified chart region is visible.

Signature: C#

```
public boolean Visible{ get; set; }
```

Signature: VB.NET

```
Public Property Visible() As Boolean
```

ChartLine

Description

A ChartLine object represents a line in a chart.

Signature: C#

```
public sealed class ChartLine
```

Signature: VB.NET

```
Public NotInheritable Class ChartLine
```

Properties

Name	Description
Color	Sets or returns the color of the line.
Style	Sets or returns the line's style.
Visible	Sets or returns whether the line is visible or hidden.
Weight	Sets or returns the weight of the line.

Nested Classes

Name	Description
LineStyle	The LineStyle class contains all available chart line styles.
LineWeight	A LineWeight value determines how wide the specified line will be.

ChartLine.Color

Description

Sets or returns the color of the line.

Signature: C#

```
public Color Color{ get; set; }
```

Signature: VB.NET

```
Public Property Color() As Color
```

ChartLine.LineStyle

Description

The LineStyle class contains all available chart line styles.

Signature: C#

```
public sealed class LineStyle : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class LineStyle
  Inherits TypesafeEnum
```

Values

Name	Description
DarkGray	Dark gray line
Dash	Dashed line
DashDot	Dash-dot line
DashDotDot	Dash-dot-dot line
Dot	Dotted line
LightGray	Light gray line
MediumGray	Medium gray line
None	No line
Solid	Solid line

ChartLine.LineWeight

Description

A LineWeight value determines how wide the specified line will be.

Signature: C#

```
public sealed class LineWeight : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class LineWeight
  Inherits TypesafeEnum
```

Values

Name	Description
Hair	Very thin line
Medium	Medium line
Narrow	Thin line
Wide	Bold line

ChartLine.Style

Description

Sets or returns the line's style.

Signature: C#

```
public LineStyle Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As LineStyle
```

ChartLine.Visible

Description

Sets or returns whether the line is visible or hidden.

Signature: C#

```
public boolean Visible{ get; set; }
```

Signature: VB.NET

```
Public Property Visible() As Boolean
```

ChartLine.Weight

Description

Sets or returns the weight of the line.

Signature: C#

```
public LineWeight Weight{ get; set; }
```

Signature: VB.NET

```
Public Property Weight() As LineWeight
```

ChartPageSetup

Description

The ChartPageSetup class represents the page layout properties of a printed chart or chart sheet. To return a `ChartPageSetup` object, use Chart.PageSetup

Signature: C#

```
public sealed class ChartPageSetup : PageSetup
```

Signature: VB.NET

```
Public NotInheritable Class ChartPageSetup
  Inherits PageSetup
```

Remarks

Note: Since this object is tightly related to client's printer setup, for those computers without a printer setup, changes to this object have no effect.

Properties

Name	Description
ChartSize	Sets or returns the PrintedChartSize value for this char. This property affects how you can move and size the chart area, plot area, and the chart elements when printed.

Nested Classes

Name	Description
PrintedChartSize	A chart's `PrintedChartSize` value determines how you can move and size the chart area, plot area, and the chart elements when printing the chart.

ChartPageSetup.ChartSize

Description

Sets or returns the PrintedChartSize value for this char. This property affects how you can move and size the chart area, plot area, and the chart elements when printed.

Signature: C#

```
public PrintedChartSize ChartSize{ get; set; }
```

Signature: VB.NET

```
Public Property ChartSize() As PrintedChartSize
```

Remarks

If you set this to PrintedChartSize.UseFullPage, the chart's plot area and elements contained within the plot area can be sized and moved, but not the chart area. When you select the PrintedChartSize.ScaleToFitPage or PrintedChartSize.Custom, the entire chart can be sized. With either of these set, the positions of chart elements such as titles, text boxes, and arrows shift relative to the chart area.

ChartPageSetup.PrintedChartSize

Description

A chart's `PrintedChartSize` value determines how you can move and size the chart area, plot area, and the chart elements when printing the chart.

Signature: C#

```
public sealed class PrintedChartSize : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class PrintedChartSize
  Inherits TypesafeEnum
```

Values

Name	Description
Custom	The entire chart (the chart area) can be sized and moved. The positions of chart elements such as titles, text boxes, and arrows shift relative to the chart area.
ScaleToFitPage	The entire chart (the chart area) can be sized and moved. The positions of chart elements such as titles, text boxes, and arrows shift relative to the chart area.
UseFullPage	The chart's plot area and elements within the plot area can be sized and moved, but not the chart area.

ChartRegion

Description

A ChartRegion object represents either a chart area or plot area. The following classes extend `ChartRegion` : AxisLabel, ChartText, Legend, DataPoint, and SeriesDataLabel.

Signature: C#

```
public class ChartRegion
```

Signature: VB.NET

```
Public Class ChartRegion
```

Properties

Name	Description
BorderLine	Returns the border around the chart region.
Font	Sets or returns the font for text in the chart region. If the ChartRegion object represents the chart area, the font returned will be the default font for all text items outside of the plot area. If this `ChartRegion` represents the plot area, the font returned will be the default font for all text items within the plot area.
HasShadow	Sets or returns whether the chart region will have a shadow.
Height	Sets or returns the height of the chart region, in units of 1/4000 of the chart area.
Interior	Returns a ChartInterior object containing formatting properties for the region.
TextDisplayTopToBottom	Sets or returns whether the text in the ChartRegion is displayed top to bottom and the letters are upright. If the ChartRegion object represents the chart area, the value returned will be the default for all text items outside of the plot area. If the `ChartRegion` object represents the plot area, the value returned will be the default for all text items within the plot area.
TextHorizontalAlignment	Sets or returns the horizontal alignment of text in the chart region. If the ChartRegion object represents the chart area, the horizontal alignment returned will be the default horizontal alignment for all text items outside the plot area. If the `ChartRegion` object represents the plot area, the horizontal alignment returned will be the default horizontal alignment for all text items within the plot area.
TextRotationAngle	Sets or returns the rotation angle of text in a chart region. If the ChartRegion object represents the chart area, the angle returned will be the default rotation angle for all text items outside of the plot area. If the `ChartRegion` object represents the plot area, the angle returned will be the default rotation angle for all text items within the plot area.
TextVerticalAlignment	Sets or returns the vertical alignment of text in the chart region. If the ChartRegion object represents the chart area, the vertical alignment returned will be the default vertical alignment for all text items outside the plot area. If the `ChartRegion` object represents the plot area, the vertical alignment returned will be the default vertical alignment for all text items within the plot area.
Width	Sets or returns the width of the chart region, in units of 1/4000 of the chart area.
X	Sets or returns the x coordinate of the upper left corner of the chart region, in units of 1/4000 of the chart area.
Y	Sets or returns the y coordinate of the upper left corner of the chart region, in units of 1/4000 of the chart area.

Nested Classes

Name	Description
HorizontalAlignment	A ChartRegion.HorizontalAlignment value determines the horizontal alignment of text in the specified chart area or plot area.
VerticalAlignment	A ChartRegion.VerticalAlignment value determines the vertical alignment of text in the specified chart area or plot area.

ChartRegion.BorderLine

Description

Returns the border around the chart region.

Signature: C#

```
public ChartLine BorderLine{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property BorderLine() As ChartLine
```

ChartRegion.Font

Description

Sets or returns the font for text in the chart region. If the ChartRegion object represents the chart area, the font returned will be the default font for all text items outside of the plot area. If this `ChartRegion` represents the plot area, the font returned will be the default font for all text items within the plot area.

Signature: C#

```
public Font Font{ get; set; }
```

Signature: VB.NET

```
Public Property Font() As Font
```

ChartRegion.HasShadow

Description

Sets or returns whether the chart region will have a shadow.

Signature: C#

```
public boolean HasShadow{ get; set; }
```

Signature: VB.NET

```
Public Property HasShadow() As Boolean
```

ChartRegion.Height

Description

Sets or returns the height of the chart region, in units of 1/4000 of the chart area.

Signature: C#

```
public int Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Integer
```

ChartRegion.HorizontalAlignment

Description

A ChartRegion.HorizontalAlignment value determines the horizontal alignment of text in the specified chart area or plot area.

Signature: C#

```
public sealed class HorizontalAlignment : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class HorizontalAlignment
  Inherits TypesafeEnum
```

Values

Name	Description
Center	Text in the chart region will be horizontally centered.
Justify	Text in the chart region will be justified.
Left	Text in the chart region will be left-aligned.
Right	Text in the chart region will be right-aligned.

ChartRegion.Interior

Description

Returns a ChartInterior object containing formatting properties for the region.

Signature: C#

```
public ChartInterior Interior{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Interior() As ChartInterior
```

ChartRegion.TextDisplayTopToBottom

Description

Sets or returns whether the text in the ChartRegion is displayed top to bottom and the letters are upright. If the ChartRegion object represents the chart area, the value returned will be the default for all text items outside of the plot area. If the `ChartRegion` object represents the plot area, the value returned will be the default for all text items within the plot area.

Signature: C#

```
public boolean TextDisplayTopToBottom{ get; set; }
```

Signature: VB.NET

```
Public Property TextDisplayTopToBottom() As Boolean
```

ChartRegion.TextHorizontalAlignment

Description

Sets or returns the horizontal alignment of text in the chart region. If the ChartRegion object represents the chart area, the horizontal alignment returned will be the default horizontal alignment for all text items outside the plot area. If the `ChartRegion` object represents the plot area, the

horizontal alignment returned will be the default horizontal alignment for all text items within the plot area.

Signature: C#

```
public HorizontalAlignment TextHorizontalAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property TextHorizontalAlignment() As HorizontalAlignment
```

ChartRegion.TextRotationAngle

Description

Sets or returns the rotation angle of text in a chart region. If the ChartRegion object represents the chart area, the angle returned will be the default rotation angle for all text items outside of the plot area. If the `ChartRegion` object represents the plot area, the angle returned will be the default rotation angle for all text items within the plot area.

Signature: C#

```
public int TextRotationAngle{ get; set; }
```

Signature: VB.NET

```
Public Property TextRotationAngle() As Integer
```

ChartRegion.TextVerticalAlignment

Description

Sets or returns the vertical alignment of text in the chart region. If the ChartRegion object represents the chart area, the vertical alignment returned will be the default vertical alignment for all text items outside the plot area. If the `ChartRegion` object represents the plot area, the vertical alignment returned will be the default vertical alignment for all text items within the plot area.

Signature: C#

```
public VerticalAlignment TextVerticalAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property TextVerticalAlignment() As VerticalAlignment
```

ChartRegion.VerticalAlignment

Description

A ChartRegion.VerticalAlignment value determines the vertical alignment of text in the specified chart area or plot area.

Signature: C#

```
public sealed class VerticalAlignment : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class VerticalAlignment
  Inherits TypesafeEnum
```

Values

Name	Description
Bottom	Text in the chart region will be bottom-aligned.
Center	Text in the chart region will be vertically centered.
Justify	Text in the chart region will be justified vertically.
Top	Text in the chart region will be top-aligned.

ChartRegion.Width

Description

Sets or returns the width of the chart region, in units of 1/4000 of the chart area.

Signature: C#

```
public int Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Integer
```

ChartRegion.X

Description

Sets or returns the x coordinate of the upper left corner of the chart region, in units of 1/4000 of the chart area.

Signature: C#

```
public int X{ get; set; }
```

Signature: VB.NET

```
Public Property X() As Integer
```

Remarks

For SeriesDataLabel s on a column or bar chart X is the offset perpendicular to the bar or column from the default position in 1/4000 of the ChartArea. For `SeriesDataLabel` s on a pie chart X is offset angle from the default position in radians. For `SeriesDataLabel` s in all other charts types X is the offset from the default x position in 1/4000 of the ChartArea.

ChartRegion.Y

Description

Sets or returns the y coordinate of the upper left corner of the chart region, in units of 1/4000 of the chart area.

Signature: C#

```
public int Y{ get; set; }
```

Signature: VB.NET

```
Public Property Y() As Integer
```

Charts

Description

The Charts collection contains all charts in the specified worksheet. Chart sheets are not included in this collection. To return a `Charts` collection, use Worksheet.Charts.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Charts : System.Collections.Generic.IEnumerable<Chart>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Charts
  Implements System.Collections.Generic.IEnumerable(Of Chart)
```

Properties

Name	Description
Count	Returns the number of charts in the worksheet.

Indexers

Name	Description
Item(Int32)	Returns the chart at the specified 0-based position in the Charts collection.
Item(String)	Returns the chart with the specified name in the Charts collection.

Methods

Name	Description
AddChart(Chart, Anchor)	Adds the specified chart to the current worksheet. This method allows you to add a chart from an external workbook to the worksheet. The chart will be auto-fitted to the default size.
CreateChart(ChartType, Anchor)	Creates a new, blank chart of the specified type.
GetEnumerator()	Returns an IEnumerator for the Charts in a Worksheet
Remove(Chart)	Permanently removes the specified chart from the worksheet.
Remove(Int32)	Permanently removes the chart at the specified position from the worksheet.

Charts.AddChart(Chart, Anchor)

Description

Adds the specified chart to the current worksheet. This method allows you to add a chart from an external workbook to the worksheet. The chart will be auto-fitted to the default size.

Signature: C#

```
public void AddChart(Chart chart, Anchor cornerTopLeft)
```

Signature: VB.NET

```
Public Sub AddChart(ByVal chart As Chart, ByVal cornerTopLeft As Anchor)
```

Parameters

chart

The new Chart to add to the workbook.

cornerTopLeft

The place to anchor the upper left corner.

Charts.Count

Description

Returns the number of charts in the worksheet.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Charts.CreateChart(ChartType, Anchor)

Description

Creates a new, blank chart of the specified type.

Signature: C#

```
public Chart CreateChart(ChartType type, Anchor cornerTopLeft)
```

Signature: VB.NET

```
Public Function CreateChart(ByVal type As ChartType, ByVal cornerTopLeft As Anchor) As Chart
```

Parameters

type

The type of chart to create.

cornerTopLeft

An Anchor representing the top left corner of the chart's position.

Returns

a new Chart.

Charts.GetEnumerator()

Description

Returns an IEnumerator for the Charts in a Worksheet

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the Comments in a Worksheet as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = ws.Charts.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = ws.Charts.GetEnumerator()
```

Charts.Item

Overload	Description
Item(Int32)	Returns the chart at the specified 0-based position in the Charts collection.
Item(String)	Returns the chart at the specified 0-based position in the Charts collection.

Charts.Item(Int32)

Description

Returns the chart at the specified 0-based position in the Charts collection.

Signature: C#

```
public Chart this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As Chart
```

Parameters

index

The 0-based index of the chart to retrieve. To determine a chart's index, use Chart.Position.

Returns

The chart at the specified index.

Charts.Item(String)

Description

Returns the chart with the specified name in the Charts collection.

Signature: C#

```
public Chart this[System.String name] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal name As String) As Chart
```

Parameters

name

The name of the chart to retrieve

Returns

The chart with the specified name.

Charts.Remove

Overload	Description
Remove(Chart)	Permanently removes the specified chart from the worksheet.
Remove(Int32)	Permanently removes the chart at the specified position from the worksheet.

Charts.Remove(Chart)

Description

Permanently removes the specified chart from the worksheet.

Signature: C#

```
public void Remove(Chart chart)
```

Signature: VB.NET

```
Public Sub Remove(ByVal chart As Chart)
```

Parameters

Chart

A Chart object representing the Chart to remove.

Charts.Remove(Int32)

Description

Permanently removes the chart at the specified position from the worksheet.

Signature: C#

```
public void Remove(int position)
```

Signature: VB.NET

```
Public Sub Remove(ByVal position As Integer)
```

Parameters

position

The 0-based index of the chart to remove. To determine a chart's index, use Chart.Position.

ChartSettings3D

Description

The ChartSettings3D class contains the set of properties specific to three-dimensional charts. To return a `ChartSettings3D` object, use Chart.ChartSettings3D.

Signature: C#

```
public sealed class ChartSettings3D
```

Signature: VB.NET

```
Public NotInheritable Class ChartSettings3D
```

Properties

Name	Description
ChartDepth	Sets or returns the depth of a three-dimensional chart.
DistanceFromEyeToChart	Sets or returns a perspective value, between 0 and 100 degrees, that represents the ratio of the chart's front to its back. The perspective value determines the appearance of chart depth.
ElevationAngle	Sets or returns the elevation angle of a three-dimensional chart.
Floor3D	Returns a ChartInterior object representing the floor of a three-dimensional chart.
Floor3DBorder	Returns a ChartLine object representing the border around the floor of a three-dimensional chart.
PlotVolumeHeight	Sets or returns the height of a three-dimensional chart, as a percentage of the chart's base length.
RotationAngle	Sets or returns the rotation angle of a three-dimensional chart.
SeriesGapWidth	Sets or returns the gap width between categories of series values.
UseAutoScaling	Sets or returns whether Microsoft Excel will automatically scale a three-dimensional chart to be close to the size of its two-dimensional version.
UseRightAngleAxes	Sets or returns whether the axes will be at right angles. If the method returns `true`, `setDistanceFromEyeToChart` will have no effect.
Wall3D	Returns an ChartInterior object representing the walls of a three-dimensional chart.
Wall3DBorder	Returns a ChartLine object representing the border around the walls of a three-dimensional chart.

ChartSettings3D.ChartDepth

Description

Sets or returns the depth of a three-dimensional chart.

Signature: C#

```
public int ChartDepth{ get; set; }
```

Signature: VB.NET

```
Public Property ChartDepth() As Integer
```

ChartSettings3D.DistanceFromEyeToChart

Description

Sets or returns a perspective value, between 0 and 100 degrees, that represents the ratio of the chart's front to its back. The perspective value determines the appearance of chart depth.

Signature: C#

```
public int DistanceFromEyeToChart{ get; set; }
```

Signature: VB.NET

```
Public Property DistanceFromEyeToChart() As Integer
```

ChartSettings3D.ElevationAngle

Description

Sets or returns the elevation angle of a three-dimensional chart.

Signature: C#

```
public int ElevationAngle{ get; set; }
```

Signature: VB.NET

```
Public Property ElevationAngle() As Integer
```

ChartSettings3D.Floor3D

Description

Returns a ChartInterior object representing the floor of a three-dimensional chart.

Signature: C#

```
public ChartInterior Floor3D{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Floor3D() As ChartInterior
```

ChartSettings3D.Floor3DBorder

Description

Returns a ChartLine object representing the border around the floor of a three-dimensional chart.

Signature: C#

```
public ChartLine Floor3DBorder{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Floor3DBorder() As ChartLine
```

ChartSettings3D.PlotVolumeHeight

Description

Sets or returns the height of a three-dimensional chart, as a percentage of the chart's base length.

Signature: C#

```
public int PlotVolumeHeight{ get; set; }
```

Signature: VB.NET

```
Public Property PlotVolumeHeight() As Integer
```

ChartSettings3D.RotationAngle

Description

Sets or returns the rotation angle of a three-dimensional chart.

Signature: C#

```
public int RotationAngle{ get; set; }
```

Signature: VB.NET

```
Public Property RotationAngle() As Integer
```

ChartSettings3D.SeriesGapWidth

Description

Sets or returns the gap width between categories of series values.

Signature: C#

```
public int SeriesGapWidth{ get; set; }
```

Signature: VB.NET

```
Public Property SeriesGapWidth() As Integer
```

ChartSettings3D.UseAutoScaling

Description

Sets or returns whether Microsoft Excel will automatically scale a three-dimensional chart to be close to the size of its two-dimensional version.

Signature: C#

```
public boolean UseAutoScaling{ get; set; }
```

Signature: VB.NET

```
Public Property UseAutoScaling() As Boolean
```

Remarks

Enabling UseAutoScaling will have no effect unless UseRightAngleAxes is set to `true`.

ChartSettings3D.UseRightAngleAxes

Description

Sets or returns whether the axes will be at right angles. If the method returns `true`, `setDistanceFromEyeToChart` will have no effect.

Signature: C#

```
public boolean UseRightAngleAxes{ get; set; }
```

Signature: VB.NET

```
Public Property UseRightAngleAxes() As Boolean
```

ChartSettings3D.Wall3D

Description

Returns an ChartInterior object representing the walls of a three-dimensional chart.

Signature: C#

```
public ChartInterior Wall3D{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Wall3D() As ChartInterior
```

ChartSettings3D.Wall3DBorder

Description

Returns a ChartLine object representing the border around the walls of a three-dimensional chart.

Signature: C#

```
public ChartLine Wall3DBorder{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Wall3DBorder() As ChartLine
```

Chartsheet

Description

A Chartsheet object represents a worksheet that contains only a chart.

Signature: C#

```
public sealed class Chartsheet : Worksheet
```

Signature: VB.NET

```
Public NotInheritable Class Chartsheet
  Inherits Worksheet
```

Remarks

To create a `Chartsheet`, use Worksheets.createChartsheet(). To get an existing `Chartsheet`, get a Worksheet using Workbook.Worksheets[i] and check if it is of type `Chartsheet`.

Examples

C#

```
//--- Create a Chartsheet
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Chartsheet cs = wb.Worksheets.CreateChartsheet
               (ChartType.Pie.Pie3D, "Chart");

          //--- Get the first Chartsheet from a Workbook
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Open("C:\\MySpreadsheet.xls");
          bool found = false;
          for(int i = 0; i < wb.Worksheets.Count; i++)
          {
               if (found == false)
               {
                    Worksheet ws = wb.Worksheets[i];

                    if(ws is Chartsheet)
                    {
                         Chartsheet cs = (Chartsheet)ws;
                         found = true;
                    }
               }
          }
```

vb.net

```
'--- Create a Chartsheet
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim cs As Chartsheet = wb.Worksheets.CreateChartsheet _
               (ChartType.Pie.Pie3D, "Chart")

          '--- Get the first Chartsheet from a Workbook
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Open("C:\MySpreadsheet.xls")
          Dim found As Boolean = False
          Dim i As Integer
          For i = 0 To wb.Worksheets.Count - 1
               If found = False Then
                    Dim ws As Worksheet = wb.Worksheets(i)

                    If ws Is Chartsheet Then
                         Dim cs As Chartsheet = CType(ws, Chartsheet)
                         found = True
                    End If
               End If
          Next
```

Properties

Name	Description
Chart	Returns a Chart object representing the chart in the chart sheet.

Chartsheet.Chart

Description

Returns a Chart object representing the chart in the chart sheet.

Signature: C#

```
public Chart Chart{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Chart() As Chart
```

Examples

C#

```
Chart c = cs.Chart;
```

vb.net

```
Dim c As Chart = cs.Chart
```

ChartText

Description

A ChartText object represents a region in a chart where text is displayed (e.g., a chart title). The `ChartText` extends ChartRegion.

Signature: C#

```
public class ChartText : ChartRegion
```

Signature: VB.NET

```
Public Class ChartText
  Inherits ChartRegion
```

Properties

Name	Description
Formula	Sets or returns the formula to use to calculate the value for the text item. For example, if the formula is "=Sheet1!A1", the value in that cell will be the text displayed in the ChartText object.
NumberFormat	Sets or returns the display format of numbers in the ChartText object.
Text	Sets or returns the text displayed in the ChartText object.

Methods

Name	Description
ClearRichTextFormatting()	Clears all rich text formatting in the ChartText object.
GetCharacters(Int32)	Returns a CharacterRun object representing a range of characters within the text. Use this to set a font on specified characters within the ChartText object.
GetCharacters(Int32, Int32)	Returns a CharacterRun object representing a range of characters within the text. Use this to set a font on specified characters within the ChartText object.

ChartText.ClearRichTextFormatting()

Description

Clears all rich text formatting in the ChartText object.

Signature: C#

```
public void ClearRichTextFormatting()
```

Signature: VB.NET

```
Public Sub ClearRichTextFormatting()
```

ChartText.Formula

Description

Sets or returns the formula to use to calculate the value for the text item. For example, if the formula is "=Sheet1!A1", the value in that cell will be the text displayed in the ChartText object.

Signature: C#

```
public System.String Formula{ get; set; }
```

Signature: VB.NET

```
Public Property Formula() As String
```

Remarks

The formula may be a reference to a cell, range of cells, or a union of such references.

ChartText.GetCharacters

Overload	Description
GetCharacters(Int32)	Returns a CharacterRun object representing a range of characters within the text. Use this to set a font on specified characters within the ChartText object.
GetCharacters(Int32, Int32)	Returns a CharacterRun object representing a range of characters within the text. Use this to set a font on specified characters within the ChartText object.

ChartText.GetCharacters(Int32, Int32)

Description

Returns a CharacterRun object representing a range of characters within the text. Use this to set a font on specified characters within the ChartText object.

Signature: C#

```
public CharacterRun GetCharacters(int startIndex, int length)
```

Signature: VB.NET

```
Public Function GetCharacters(ByVal startIndex As Integer, ByVal length As Integer) As CharacterRun
```

Parameters

startIndex

The character position at which to begin the character run.

length

The length of the character run.

Returns

A `CharacterRun` object representing a range of characters within the cell.

ChartText.GetCharacters(Int32)

Description

Returns a CharacterRun object representing a range of characters within the text. Use this to set a font on specified characters within the ChartText object.

Signature: C#

```
public CharacterRun GetCharacters(int startIndex)
```

Signature: VB.NET

```
Public Function GetCharacters(ByVal startIndex As Integer) As CharacterRun
```

Parameters

startIndex

The 0-based character position at which to begin the character run.

Returns

A `CharacterRun` object representing a range of characters within the text.

ChartText.NumberFormat

Description

Sets or returns the display format of numbers in the ChartText object.

Signature: C#

```
public System.String NumberFormat{ get; set; }
```

Signature: VB.NET

```
Public Property NumberFormat() As String
```

ChartText.Text

Description

Sets or returns the text displayed in the ChartText object.

Signature: C#

```
public System.String Text{ get; set; }
```

Signature: VB.NET

```
Public Property Text() As String
```

ChartTextCollection

Description

The ChartTextCollection contains all ChartText objects in the specified chart..

Signature: C#

```
[DefaultMember("Item")]
 public abstract interface ChartTextCollection
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public MustInherit Interface ChartTextCollection
 End Interface
```

Properties

Name	Description
Count	Returns number of ChartText objects in the collection.

Indexers

Name	Description
Item(Int32)	Returns the ChartText object at the specified index.

Methods

Name	Description
AddChartText(ChartText)	Adds an existing ChartText object to this ChartTextCollection.
CreateChartText()	Create a new blank ChartText object which is automatically added to the ChartTextCollection.
CreateChartText(String)	Create a new ChartText object with the specified string as its text. The new object is automatically added to the ChartTextCollection.
Remove(ChartText)	Permanently removes a ChartText object from the collection.
Remove(Int32)	Permanently removes a ChartText object from the collection.

ChartTextCollection.AddChartText(ChartText)

Description

Adds an existing ChartText object to this ChartTextCollection.

Signature: C#

```
public abstract void AddChartText(ChartText text)
```

Signature: VB.NET

```
Public MustOverride Sub AddChartText(ByVal text As ChartText)
```

Parameters

text

The `ChartText` object to add.

ChartTextCollection.Count

Description

Returns number of ChartText objects in the collection.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

ChartTextCollection.CreateChartText

Overload	Description
CreateChartText()	Create a new blank ChartText object which is automatically added to the ChartTextCollection.
CreateChartText(String)	Create a new ChartText object with the specified string as its text. The new object is automatically added to the ChartTextCollection.

ChartTextCollection.CreateChartText()

Description

Create a new blank ChartText object which is automatically added to the ChartTextCollection.

Signature: C#

```
public abstract ChartText CreateChartText()
```

Signature: VB.NET

```
Public MustOverride Function CreateChartText() As ChartText
```

Returns

A new `ChartText` object.

ChartTextCollection.CreateChartText(String)

Description

Create a new ChartText object with the specified string as its text. The new object is automatically added to the ChartTextCollection.

Signature: C#

```
public abstract ChartText CreateChartText(System.String text)
```

Signature: VB.NET

```
Public MustOverride Function CreateChartText(ByVal text As String) As ChartText
```

Parameters

text

The text to display in the new `ChartText` object.

Returns

A new `ChartText` object.

ChartTextCollection.Item(Int32)

Description

Returns the ChartText object at the specified index.

Signature: C#

```
public ChartText this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As ChartText
```

Parameters

index

The 0-based index of the `ChartText` to retrieve.

Returns

The `ChartText` object at the specified index.

Remarks

The `ChartText` property is the indexer for the ChartTextCollection class.

ChartTextCollection.Remove

Overload	Description
Remove(ChartText)	Permanently removes a ChartText object from the collection.
Remove(Int32)	Permanently removes a ChartText object from the collection.

ChartTextCollection.Remove(ChartText)

Description

Permanently removes a ChartText object from the collection.

Signature: C#

```
public abstract void Remove(ChartText chartText)
```

Signature: VB.NET

```
Public MustOverride Sub Remove(ByVal chartText As ChartText)
```

Parameters

chartText

The `ChartText` object to remove.

ChartTextCollection.Remove(Int32)

Description

Permanently removes a ChartText object from the collection.

Signature: C#

```
public abstract void Remove(int position)
```

Signature: VB.NET

```
Public MustOverride Sub Remove(ByVal position As Integer)
```

Parameters

position

The 0-based position of the `ChartText` in the collection.

ChartType

Description

The ChartType class contains all available chart types. When setting a chart type, always specify both the major type and the sub-type, for example:

Signature: C#

```
public abstract class ChartType
```

Signature: VB.NET

```
Public MustInherit Class ChartType
```

Properties

Name	Description
Category	Returns the ChartType's chart category (Column, Pie, Line, Radar, Bar, Surface, Area, Scatter, Doughnut, Bubble, or Custom).
Is3D	Returns whether the chart type is three-dimensional or not.
Name	Returns the name of the chart type as a string.

Nested Classes

Name	Description
Area	Enumerations for area chart types
Bar	Enumerations for bar chart types
Bubble	Enumerations for Bubble chart types
ChartCategory	A ChartCategory specifies a chart type's category. For example, column is a chart category, while clustered column and stacked column are chart types in the column category.
Column	Enumerations for Column chart types
Custom	Enumerations for Customer chart types
Doughnut	Enumerations for doughnut chart types
Line	Enumerations for line chart types
Pie	Enumerations for pie chart types
Radar	The page ChartType.Radar does not exist.

Scatter	Enumerations for scatter chart types
Surface	Enumerations for surface chart types

ChartType.Area

Description

Enumerations for area chart types

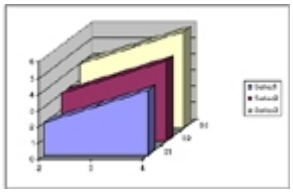
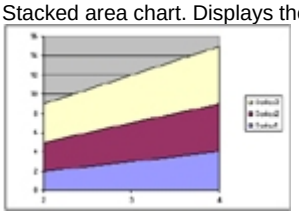
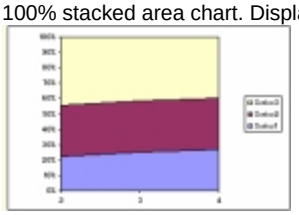
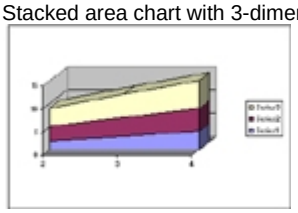
Signature: C#

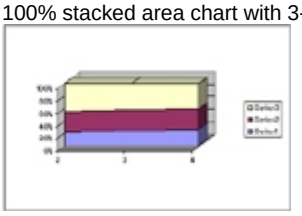
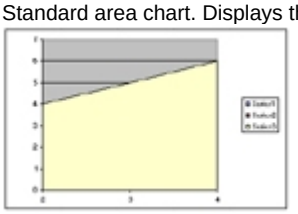
```
public sealed class Area : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Area
  Inherits ChartPrototype
```

Values

Name	Description
Area3D	Area chart with 3-dimensional visual effect. 
Stacked	Stacked area chart. Displays the trend of the contribution of each value over time or categories. 
Stacked100Percent	100% stacked area chart. Displays the trend of the percentage each value contributes over time or categories. 
Stacked3D	Stacked area chart with 3-dimensional visual effect. 

Stacked3D100Percent	100% stacked area chart with 3-dimensional visual effect. 
StandardArea	Standard area chart. Displays the trend of values over time or categories. 

ChartType.Bar

Description

Enumerations for bar chart types

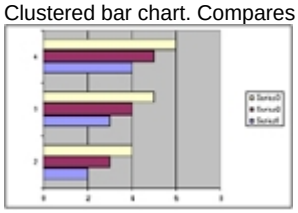
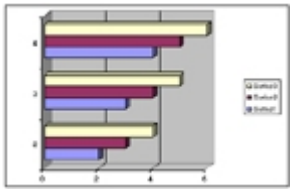
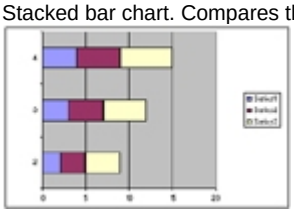
Signature: C#

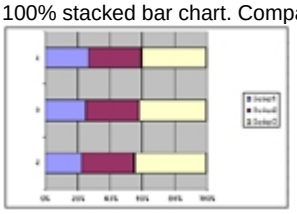
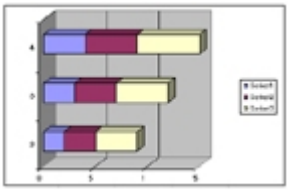
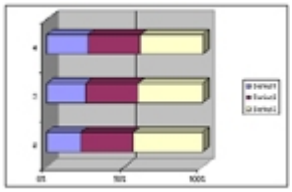
```
public sealed class Bar : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Bar
  Inherits ChartPrototype
```

Values

Name	Description
Clustered	Clustered bar chart. Compares values across categories. 
Clustered3D	Clusterd bar chart with a 3-dimensional visual effect. 
Stacked	Stacked bar chart. Compares the contribution of each value to a total across categories. 

Stacked100Percent	100% stacked bar chart. Compares the percentage each value contributes to a total across categories. 
Stacked3D	Stacked bar chart with a 3-dimensional visual effect. 
Stacked3D100Percent	100% stacked bar chart with a 3-dimensional visual effect. 

ChartType.Bubble

Description

Enumerations for Bubble chart types

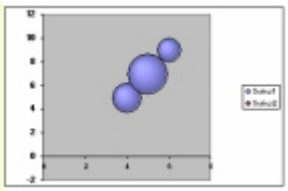
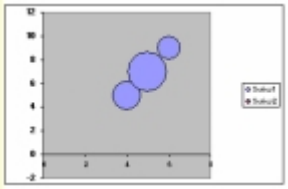
Signature: C#

```
public sealed class Bubble : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Bubble
  Inherits ChartPrototype
```

Values

Name	Description
Bubble3D	Bubble chart with a 3-dimensional visual effect. 
StandardBubble	Standard bubble chart. 

ChartType.Category

Description

Returns the ChartType's chart category (Column, Pie, Line, Radar, Bar, Surface, Area, Scatter, Doughnut, Bubble, or Custom).

Signature: C#

```
public ChartCategory Category{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Category() As ChartCategory
```

ChartType.ChartCategory

Description

A ChartCategory specifies a chart type's category. For example, column is a chart category, while clustered column and stacked column are chart types in the column category.

Signature: C#

```
public sealed class ChartCategory : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ChartCategory
  Inherits TypesafeEnum
```

Values

Name	Description
Area	Area chart
Bar	Bar chart
Bubble	Bubble chart
Column	Column chart
Custom	Custom chart
Doughnut	Doughnut chart
Line	Line chart
Pie	Pie chart
Radar	Radar chart
Scatter	Scatter chart
Surface	Surface chart

ChartType.Column

Description

Enumerations for Column chart types

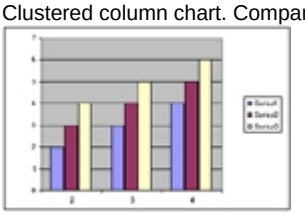
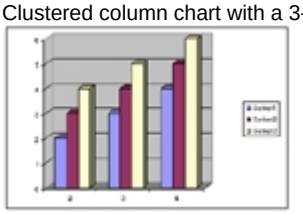
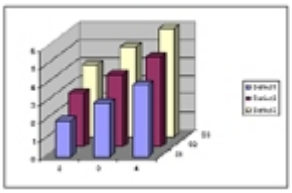
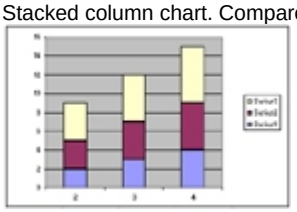
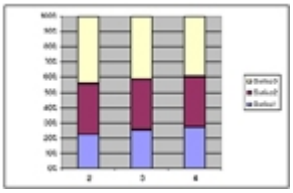
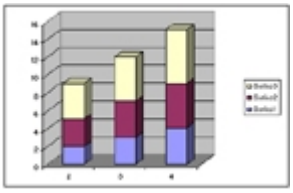
Signature: C#

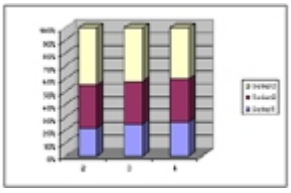
```
public sealed class Column : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Column
  Inherits ChartPrototype
```

Values

Name	Description
Clustered	Clustered column chart. Compares values across categories. 
Clustered3D	Clustered column chart with a 3-dimensional visual effect. 
Column3D	3-dimensional column chart. Compares values across categories and across series. 
Stacked	Stacked column chart. Compares the contribution of each value to a total across categories. 
Stacked100Percent	100% stacked column chart. Compares the percentage each value contributes to a total across categories. 
Stacked3D	Stacked column chart with a 3-dimensional visual effect. 

Stacked3D100Percent	100% stacked column chart with a 3-dimensional visual effect. 

ChartType.Custom

Description

Enumerations for Customer chart types

Signature: C#

```
public sealed class Custom : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class [Custom]
  Inherits ChartPrototype
```

Values

Name	Description
Combo	Custom chart type.

ChartType.Doughnut

Description

Enumerations for doughnut chart types

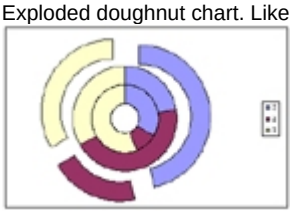
Signature: C#

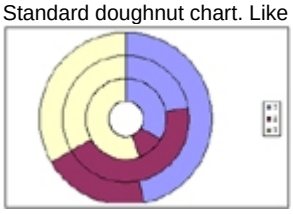
```
public sealed class Doughnut : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Doughnut
  Inherits ChartPrototype
```

Values

Name	Description
Exploded	Exploded doughnut chart. Like an exploded pie chart, but can contain multiple series. 

StandardDoughnut	Standard doughnut chart. Like a pie chart, but can contain multiple series. 

ChartType.Is3D

Description

Returns whether the chart type is three-dimensional or not.

Signature: C#

```
public boolean Is3D{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Is3D() As Boolean
```

ChartType.Line

Description

Enumerations for line chart types

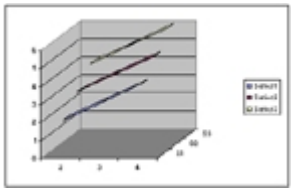
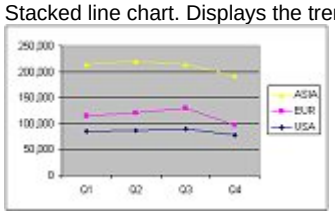
Signature: C#

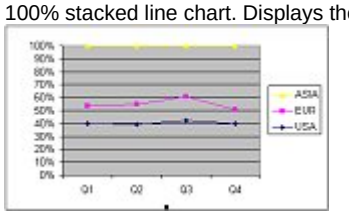
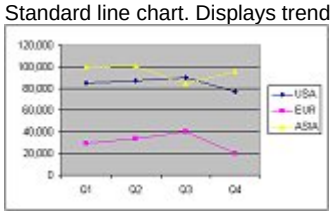
```
public sealed class Line : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Line
  Inherits ChartPrototype
```

Values

Name	Description
Line3D	Line chart with a 3-dimensional visual effect. 
Stacked	Stacked line chart. Displays the trend of the contribution of each value over time or categories. 

Stacked100Percent	100% stacked line chart. Displays the trend of the percentage each value contributes over time or categories. 
StandardLine	Standard line chart. Displays trend over time or categories. 

ChartType.Name

Description

Returns the name of the chart type as a string.

Signature: C#

```
public System.String Name{ get; }
```

Signature: VB.NET

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

ChartType.Pie

Description

Enumerations for pie chart types

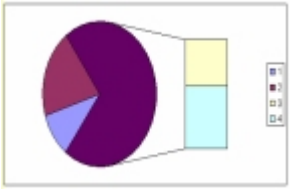
Signature: C#

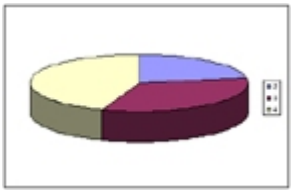
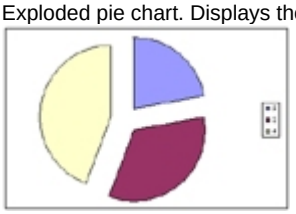
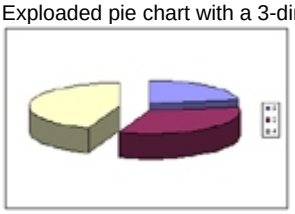
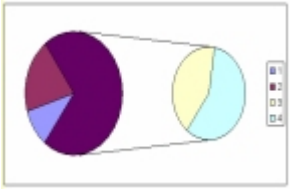
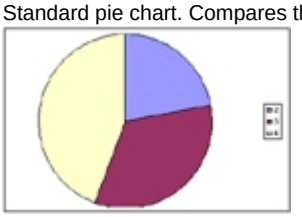
```
public sealed class Pie : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Pie
  Inherits ChartPrototype
```

Values

Name	Description
BarOfPie	Bar of pie chart. Pie with user-defined values extracted and combined into a stacked bar. 

Pie3D	Pie chart with a 3-dimensional visual effect. 
PieExploded	Exploded pie chart. Displays the contribution of each value to a total while emphasizing individual values. 
PieExploded3D	Exploaded pie chart with a 3-dimensional visual effect. 
PieOfPie	Pie of pie chart. Pie with user-defined values extracted and combined into a second pie. 
StandardPie	Standard pie chart. Compares the contribution of each value to a total. 

ChartType.Scatter

Description

Enumerations for scatter chart types

Signature: C#

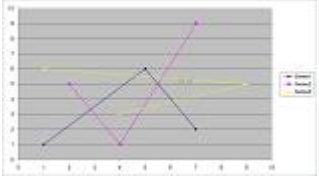
```
public sealed class Scatter : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Scatter
  Inherits ChartPrototype
```

Values

Name	Description

StandardScatter	Scatter chart. Compares pairs of values. 

ChartType.Surface

Description

Enumerations for surface chart types

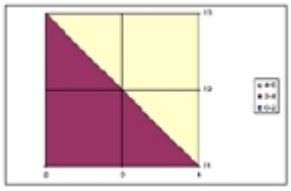
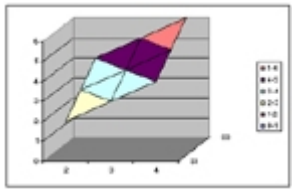
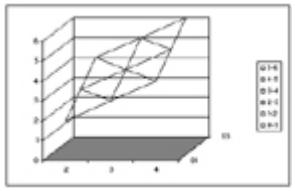
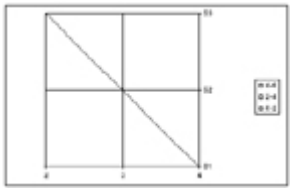
Signature: C#

```
public sealed class Surface : ChartPrototype
```

Signature: VB.NET

```
Public NotInheritable Class Surface
  Inherits ChartPrototype
```

Values

Name	Description
Contour	Contour chart. Surface chart viewed from above. 
Surface3D	3-dimensional surface chart. Shows trends in values across two dimensions in a continuous curve. 
Wireframe3D	3-dimensional surface chart without color. 
WireframeContour	Contour chart without color. 

Color

Description

The `Color` class represents a single color in a spreadsheet's color palette.

Signature: C#

```
public sealed class Color
```

Signature: VB.NET

```
Public NotInheritable Class Color
```

Remarks

Each Excel document has a single color palette that manages the colors for formatting and some drawing layer colors. A color palette contains 56 colors that may be referenced. Each of these 56 colors may be customized, but changing a color in the palette will affect all formatting that uses the referenced color. To get a `Color` object, use Workbook.Palette to get to the Workbook's palette, then use one of the Palette's methods to get a `Color`.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Palette p = wb.Palette;
SoftArtisans.OfficeWriter.ExcelApplication.Color clr =
    p.GetClosestColor(System.Drawing.Color.Blue);
```

vb.net

```
Dim xla As New ExcelApplication()
Dim wb As Workbook = xla.Create()
Dim p As Palette = wb.Palette()
Dim clr As SoftArtisans.OfficeWriter.ExcelApplication.Color = _
    p.GetClosestColor(System.Drawing.Color.Blue)
```

Properties

Name	Description
Blue	Returns the blue component of the color's RGB value.
Green	Returns the green component of color's the RGB value.
IsCustomColor	Returns whether or not the color is a custom color that is not part of Excel's default color palette.
IsPaletteColor	Returns whether or not the color is stored in Excel's default color palette.
IsSystemColor	Returns whether or not the color is a system color that cannot be changed.
PaletteIndex	Returns the 0-based index of the color in the spreadsheet's color palette.
Red	Returns the red component of the color's RGB value.

Methods

Name	Description
ToString()	Returns a string representation of the `Color` object.

Nested Classes

Name	Description
SystemColor	System colors that can be used as colors in ExcelWriter

Color.Blue

Description

Returns the blue component of the color's RGB value.

Signature: C#

```
public int Blue{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Blue() As Integer
```

Examples

C#

```
int blue = clr.Blue;
```

vb.net

```
Dim blue As Integer = clr.Blue
```

Color.Green

Description

Returns the green component of color's the RGB value.

Signature: C#

```
public int Green{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Green() As Integer
```

Examples

C#

```
int green = clr.Green;
```

vb.net

```
Dim green As Integer = clr.Green
```

Color.IsCustomColor

Description

Returns whether or not the color is a custom color that is not part of Excel's default color palette.

Signature: C#

```
public boolean IsCustomColor{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property IsCustomColor() As Boolean
```

Examples

C#

```
bool customColor = clr.IsCustomColor;
```

vb.net

```
Dim customColor As Boolean = clr.IsCustomColor
```

Color.IsPaletteColor

Description

Returns whether or not the color is stored in Excel's default color palette.

Signature: C#

```
public boolean IsPaletteColor{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property IsPaletteColor() As Boolean
```

Examples

C#

```
bool paletteColor = clr.IsPaletteColor;
```

vb.net

```
Dim paletteColor As Boolean = clr.IsPaletteColor
```

Color.IsSystemColor

Description

Returns whether or not the color is a system color that cannot be changed.

Signature: C#

```
public boolean IsSystemColor{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property IsSystemColor() As Boolean
```

Color.PaletteIndex

Description

Returns the 0-based index of the color in the spreadsheet's color palette.

Signature: C#

```
public int PaletteIndex{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PaletteIndex() As Integer
```

Examples

C#

```
int colorIndex = clr.PaletteIndex;
```

vb.net

```
Dim colorIndex As Integer = clr.PaletteIndex
```

Color.Red

Description

Returns the red component of the color's RGB value.

Signature: C#

```
public int Red{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Red() As Integer
```

Examples

C#

```
int red = clr.Red;
```

vb.net

```
Dim red As Integer = clr.Red
```

Color.SystemColor

Description

System colors that can be used as colors in ExcelWriter

Signature: C#

```
public sealed class SystemColor
```

Signature: VB.NET

```
Public NotInheritable Class SystemColor
```

Values

Name	Description
AutomaticFill	Default color for fills and backgrounds, depends on system color settings when opening the file, usually white
AutomaticLine	Default color for lines, depends on system color settings when opening the file, usually Black
Black	Black. RGB: 0,0,0; Hex: #000000
Blue	Blue. RGB: 0, 0, 255; Hex: #0000FF
Cyan	Cyan. RGB: 0, 255, 255; Hex: #00FFFF
Green	Green. RGB: 0, 255, 0; Hex: #00FF000
Magenta	Magenta. RGB: 255, 0, 255; Hex: #FF00FF
Red	Red. RGB: 255, 0, 0; Hex: #FF0000
White	White. RGB: 255, 255, 255; Hex: #FFFFFF
Yellow	Yellow. RGB: 255, 255, 0; Hex: #FFFF00

Color.ToString()

Description

Returns a string representation of the `Color` object.

Signature: C#

```
public virtual System.String ToString()
```

Signature: VB.NET

```
Public Overridable Function ToString() As String
```

Returns

A string representation of the `Color` object.

Examples

C#

```
string color = clr.ToString();
```

vb.net

```
Dim color As String = clr.ToString()
```

ColumnProperties

Description

The ColumnProperties class represents a single column in a worksheet.

Signature: C#

```
public sealed class ColumnProperties
```

Signature: VB.NET

```
Public NotInheritable Class ColumnProperties
```

Remarks

To return a `ColumnProperties` object, use Worksheet.GetColumnProperties(). The following sample gets the column properties for the first column of the first worksheet.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          ColumnProperties colProps = ws.GetColumnProperties(0);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim colProps As ColumnProperties = ws.GetColumnProperties(0)
```

Properties

Name	Description
ColumnNumber	Returns the column's 0-based index.
Hidden	Sets or returns whether or not the column will be hidden when the workbook is opened in Excel.
OutlineCollapsed	Sets or returns whether the column's outlined column group is collapsed. If the outline is collapsed, the column is hidden.
OutlineLevel	Sets or returns the outline level of the column, as a value between 0 and 7. The outline level is used to control grouping in Excel. For example, to group several columns, set their outline levels to 1; to group a subset of columns within the group, set their outline levels to 2.
PaddingInChars	Returns 'padding' (in characters) that needs to be added to the column width (in characters) to get true width of this column.
Style	Sets or returns a Style object that determines how cells in the column look, including number format, font, orientation, etc.
Width	Sets or returns the width of the column in points (1/72 of an inch).
WidthInChars	Sets or returns the width of the column as a number of '0' characters in the font of the 'Normal' style. This must be a value between 0 - 255.

Methods

Name	Description
ApplyStyle(Style)	Applies a style to the column. When a style is applied - rather than set (see ColumnProperties.Style) - only the differences between the new style and style properties previously assigned to the column (through the ExcelWriter API or in Microsoft Excel) will take effect. For example, if a cell in the column has a background color and the new style applied does not contain a background color, the cell's color will not be affected.

AutoFitWidth()	Automatically sets the width of the column to the width of the widest cell value in the column.

ColumnProperties.ApplyStyle(Style)

Description

Applies a style to the column. When a style is applied - rather than set (see ColumnProperties.Style) - only the differences between the new style and style properties previously assigned to the column (through the ExcelWriter API or in Microsoft Excel) will take effect. For example, if a cell in the column has a background color and the new style applied does not contain a background color, the cell's color will not be affected.

Signature: C#

```
public void ApplyStyle(Style style)
```

Signature: VB.NET

```
Public Sub ApplyStyle(ByVal style As Style)
```

Parameters

style

The Style object to apply to the row.

Examples

C#

```
colProps.ApplyStyle(styl);
```

vb.net

```
colProps.ApplyStyle(styl)
```

ColumnProperties.AutoFitWidth()

Description

Automatically sets the width of the column to the width of the widest cell value in the column.

Signature: C#

```
public void AutoFitWidth()
```

Signature: VB.NET

```
Public Sub AutoFitWidth()
```

Examples

C#

```
colProps.AutoFitWidth();
```

vb.net

```
colProps.AutoFitWidth()
```

ColumnProperties.ColumnNumber

Description

Returns the column's 0-based index.

Signature: C#

```
public int ColumnNumber{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ColumnNumber() As Integer
```

Examples

C#

```
int colNum = colProps.ColumnNumber;
```

vb.net

```
Dim colNum As Integer = colProps.ColumnNumber
```

ColumnProperties.Hidden

Description

Sets or returns whether or not the column will be hidden when the workbook is opened in Excel.

Signature: C#

```
public boolean Hidden{ get; set; }
```

Signature: VB.NET

```
Public Property Hidden() As Boolean
```

Examples

C#

```
bool hideCol = colProps.Hidden;
          colProps.Hidden = false;
```

vb.net

```
Dim hideCol As Boolean = colProps.Hidden
          colProps.Hidden = False
```

ColumnProperties.OutlineCollapsed

Description

Sets or returns whether the column's outlined column group is collapsed. If the outline is collapsed, the column is hidden.

Signature: C#

```
public boolean OutlineCollapsed{ get; set; }
```

Signature: VB.NET

```
Public Property OutlineCollapsed() As Boolean
```

Examples

C#

```
bool collapsed = colProps.OutlineCollapsed;
          colProps.OutlineCollapsed = true;
```

vb.net

```
Dim collapsed As Boolean = colProps.OutlineCollapsed
          colProps.OutlineCollapsed = True
```

ColumnProperties.OutlineLevel

Description

Sets or returns the outline level of the column, as a value between 0 and 7. The outline level is used to control grouping in Excel. For example, to group several columns, set their outline levels to 1; to group a subset of columns within the group, set their outline levels to 2.

Signature: C#

```
public int OutlineLevel{ get; set; }
```

Signature: VB.NET

```
Public Property OutlineLevel() As Integer
```

Examples

C#

```
int groupLevel = colProps.OutlineLevel;
          colProps.OutlineLevel = 4;
```

vb.net

```
Dim groupLevel As Integer = colProps.OutlineLevel
          colProps.OutlineLevel = 4
```

ColumnProperties.PaddingInChars

Description

Returns 'padding' (in characters) that needs to be added to the column width (in characters) to get true width of this column.

Signature: C#

```
public double PaddingInChars{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PaddingInChars() As Double
```

Remarks

For example, let's say we have ten columns that are 3.71 characters wide. What is the width of a merged cell that spans all ten columns? To find the width in characters you have to include padding 10*(WidthInChar+PaddingInChars). For Arial 10 the padding is 0.71 and the width is 44.20

ColumnProperties.Style

Description

Sets or returns a Style object that determines how cells in the column look, including number format, font, orientation, etc.

Signature: C#

```
public Style Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As Style
```

Examples

C#

```
Style styl = colProps.Style;
```

vb.net

```
Dim styl As Style = colProps.Style
```

ColumnProperties.Width

Description

Sets or returns the width of the column in points (1/72 of an inch).

Signature: C#

```
public double Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Double
```

Examples

C#

```
double colWidth = colProps.Width;
          colProps.Width = 100;
```

vb.net

```
Dim colWidth As Double = colProps.Width
          colProps.Width = 100
```

ColumnProperties.WidthInChars

Description

Sets or returns the width of the column as a number of '0' characters in the font of the 'Normal' style. This must be a value between 0 - 255.

Signature: C#

```
public double WidthInChars{ get; set; }
```

Signature: VB.NET

```
Public Property WidthInChars() As Double
```

Remarks

While a cell can show more than 255 characters, a column may not exceed 255 characters in width.

Examples

C#

```
double colWidth = colProps.WidthInChars;
          colProps.WidthInChars = 20;
```

vb.net

```
Dim colWidth As Double = colProps.WidthInChars
          colProps.WidthInChars = 20
```

Comment

Description

A Comment object represents a note attached to a cell, or group of merged cells, separate from other cell content. A `Comment` must be associated with a cell, and may only be created using set Comment.Visible to `true`.

Signature: C#

```
public sealed class Comment : Shape
```

Signature: VB.NET

```
Public NotInheritable Class Comment
  Inherits Shape
```

Properties

Name	Description
Author	Sets or returns the author of the comment. This information is displayed in the status bar in Excel when a comment is selected.
Cell	This returns the Cell object that this comment is associated with.
FitToText	Sets or returns whether the comment's shape will be automatically sized to fit the text.
HorizontalAlignment	Sets or returns the horizontal alignment of text within the shape.

Shape	Return a Shape object representing the display area of the comment. The `Shape` object may be used to set advanced formatting options such as fill, border, size, or position.
Text	Sets or returns the body text of the comment.
VerticalAlignment	Sets or returns the vertical alignment of text within the shape.
Visible	Sets or returns whether the comment will be visible when the document is opened in Excel. By default, the comment will be hidden, and will be displayed only if the user hovers over the comment's cell.

Methods

Name	Description
ClearRichTextFormatting()	Clears all rich text formatting within the comment. The font of the comment text will be set to the font set (through the ExcelWriter API) for the cell the comment is attached to.
GetCharacters(Int32)	Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.
GetCharacters(Int32, Int32)	Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.

Comment.Author

Description

Sets or returns the author of the comment. This information is displayed in the status bar in Excel when a comment is selected.

Signature: C#

```
public System.String Author{ get; set; }
```

Signature: VB.NET

```
Public Property Author() As String
```

Comment.Cell

Description

This returns the Cell object that this comment is associated with.

Signature: C#

```
public Cell Cell{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Cell() As Cell
```

Comment.ClearRichTextFormatting()

Description

Clears all rich text formatting within the comment. The font of the comment text will be set to the font set (through the ExcelWriter API) for the cell the comment is attached to.

Signature: C#

```
public virtual void ClearRichTextFormatting()
```

Signature: VB.NET

```
Public Overridable Sub ClearRichTextFormatting()
```

Comment.FitToText

Description

Sets or returns whether the comment's shape will be automatically sized to fit the text.

Signature: C#

```
public boolean FitToText{ get; set; }
```

Signature: VB.NET

```
Public Property FitToText() As Boolean
```

Comment.GetCharacters

Overload	Description
GetCharacters(Int32)	Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.
GetCharacters(Int32, Int32)	Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.

Comment.GetCharacters(Int32, Int32)

Description

Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.

Signature: C#

```
public virtual CharacterRun GetCharacters(int startIndex, int length)
```

Signature: VB.NET

```
Public Overridable Function GetCharacters(ByVal startIndex As Integer, ByVal length As Integer) As
CharacterRun
```

Parameters

startIndex

The character position at which to begin the character run.

length

The length of the character run.

Returns

A `CharacterRun` object representing a range of characters within the cell.

Comment.GetCharacters(Int32)

Description

Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within

the text.

Signature: C#

```
public virtual CharacterRun GetCharacters(int startIndex)
```

Signature: VB.NET

```
Public Overridable Function GetCharacters(ByVal startIndex As Integer) As CharacterRun
```

Parameters

startIndex

The 0-based character position at which to begin the character run.

Returns

A `CharacterRun` object representing a range of characters within the comment text.

Comment.HorizontalAlignment

Description

Sets or returns the horizontal alignment of text within the shape.

Signature: C#

```
public HAlign HorizontalAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property HorizontalAlignment() As HAlign
```

Comment.Shape

Description

Return a Shape object representing the display area of the comment. The `Shape` object may be used to set advanced formatting options such as fill, border, size, or position.

Signature: C#

```
public Shape Shape{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Shape() As Shape
```

Comment.Text

Description

Sets or returns the body text of the comment.

Signature: C#

```
public System.String Text{ get; set; }
```

Signature: VB.NET

```
Public Property Text() As String
```

Comment.VerticalAlignment

Description

Sets or returns the vertical alignment of text within the shape.

Signature: C#

```
public VAlign VerticalAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property VerticalAlignment() As VAlign
```

Comment.Visible

Description

Sets or returns whether the comment will be visible when the document is opened in Excel. By default, the comment will be hidden, and will be displayed only if the user hovers over the comment's cell.

Signature: C#

```
public boolean Visible{ get; set; }
```

Signature: VB.NET

```
Public Property Visible() As Boolean
```

Comments

Description

The Comments collection contains all Comment objects in the worksheet. To return a `Comments` collection, use Worksheet.Comments.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Comments : System.Collections.Generic.IEnumerable<Comment>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Comments
  Implements System.Collections.Generic.IEnumerable(Of Comment)
```

Properties

Name	Description
Count	Returns the number of Comment objects in the collection.

Indexers

Name	Description
Item(Int32)	Returns the Comment object at the specified position in the Comments collection.
Item(String)	Returns the Comment object with the specified name in the Comments collection.

Methods

Name	Description
CreateComment(Cell)	Creates a comment and associates it with the specified Cell.
CreateComment(Int32, Int32)	Creates a comment and associates it with the cell with the specified row and column coordinates.
GetEnumerator()	Returns an IEnumerator for the Comments in a Worksheet
Remove(Int32)	Removes the Comment object at the specified position from the collection.

Comments.Count

Description

Returns the number of Comment objects in the collection.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Comments.CreateComment

Overload	Description
CreateComment(Cell)	Creates a comment and associates it with the specified Cell.
CreateComment(Int32, Int32)	Creates a comment and associates it with the cell with the specified row and column coordinates.

Comments.CreateComment(Cell)

Description

Creates a comment and associates it with the specified Cell.

Signature: C#

```
public Comment CreateComment(Cell cell)
```

Signature: VB.NET

```
Public Function CreateComment(ByVal cell As Cell) As Comment
```

Parameters

cell

The cell to associate the comment with.

Returns

The new comment.

Comments.CreateComment(Int32, Int32)

Description

Creates a comment and associates it with the cell with the specified row and column coordinates.

Signature: C#

```
public Comment CreateComment(int row, int column)
```

Signature: VB.NET

```
Public Function CreateComment(ByVal row As Integer, ByVal column As Integer) As Comment
```

Parameters

row

The row for the new comment (0-based).

column

The column for the new comment (0-based).

Returns

The new comment.

Comments.GetEnumerator()

Description

Returns an IEnumerator for the Comments in a Worksheet

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the Comments in a Worksheet as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = ws.Comments.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = ws.Comments.GetEnumerator()
```

Comments.Item

Overload	Description

Item(Int32)	Returns the Comment object at the specified position in the Comments collection.
Item(String)	Returns the Comment object with the specified name in the Comments collection.

Comments.Item(Int32)

Description

Returns the Comment object at the specified position in the Comments collection.

Signature: C#

```
public Comment this[int position] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal position As Integer) As Comment
```

Parameters

position

A valid index between 0 and Comments.Count.

Returns

The `Comment` object at the specified position.

Comments.Item(String)

Description

Returns the Comment object with the specified name in the Comments collection.

Signature: C#

```
public Comment this[System.String name] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal name As String) As Comment
```

Parameters

position

A valid name of a `Comment` in the `Comments` collection.

Returns

The `Comment` object with the specified name.

Comments.Remove(Int32)

Description

Removes the Comment object at the specified position from the collection.

Signature: C#

```
public void Remove(int position)
```

Signature: VB.NET

```
Public Sub Remove(ByVal position As Integer)
```

Parameters

position

The 0-based position of the `Comment` to remove.

Condition

Description

A Condition object represents a condition within a ConditionalFormat, a format that Excel will apply to a cell or group of cells if a specified condition is met.

Signature: C#

```
public sealed class Condition
```

Signature: VB.NET

```
Public NotInheritable Class Condition
```

Remarks

A ConditionalFormat object contains up to three `Condition` objects and a Range object representing the set of cells on which the conditions will be tested.

To create a `Condition` object, call one of the `ConditionalFormat` object's CreateCondition methods.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
Range rng = wb.Worksheets[0].CreateRange("$B$2:$F$10");
ConditionalFormat condFmt = wb.CreateConditionalFormat();
Condition cond =
     condFmt.CreateCondition(
     Condition.Comparison.CellValueGreaterThan,
     "=100");
Font fnt = cond.Style.Font;
fnt.Bold = true;
fnt.Color = wb.Palette.GetClosestColor(100, 100, 255);
rng.SetConditionalFormat(condFmt);
```

vb.net

```
Dim xla As New ExcelApplication()
Dim wb As Workbook = xla.Create()
Dim rng As Range = wb.Worksheets(0).CreateRange("$B$2:$F$10")
Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
Dim cond As Condition = _
     condFmt.CreateCondition( _
     Condition.Comparison.CellValueGreaterThan, _
     "=100")
Dim fnt As Font = cond.Style.Font
fnt.Bold = true
fnt.Color = wb.Palette.GetClosestColor(100, 100, 255)
rng.SetConditionalFormat(condFmt)
```

Properties

Name	Description
ComparisonType	Returns the Condition's Condition.Comparison value, which determines whether each cell value will be compared with one or two other values, or if a formula will be evaluated for each cell.
Formula1	If the comparison is between a cell value and two values, this is the minimum value; otherwise this is is the formula to be evaluated for the condition.
Formula2	If the value of Condition.ComparisonType is a comparison between a cell value and two values (e.g., Condition.Comparison.CellValueBetween), Formula2 sets or returns a formula whose result will be the maximum comparison value.
Style	Sets or returns the Style that will be set on conditionally formatted cells when the current Condition is met.

Methods

Name	Description
ApplyStyle(Style)	Applies the specified Style to the Condition's current Style.
ChangeCondition(Condition.Comparison, String)	Resets the current Condition's Comparison type and formula. The method takes one formulas, and therefore must take a `Comparison` that requires one formulas, such as Comparison.CellValueLessThan.
ChangeCondition(Condition.Comparison, String, String)	Resets the current Condition's Comparison type and minimum and maximum formulas. The method takes two formulas, and therefore must take a `Comparison` that requires two formulas, such as Comparison.CellValueBetween.

Nested Classes

Name	Description
Comparison	The value of Condition.Comparison determines whether the value of a conditionally formatted cell will be compared with one or two other values, or if a formula will be evaluated for each cell.

Condition.ApplyStyle(Style)

Description

Applies the specified Style to the Condition's current Style.

Signature: C#

```
public void ApplyStyle(Style style)
```

Signature: VB.NET

```
Public Sub ApplyStyle(ByVal style As Style)
```

Parameters

style

The `Style` object to
apply to the `Condition`'s current `Style`.

Remarks

A `Condition`'s `Style` is the formatting that will be set on a conditionally formatted cell if the condition is met. When a `Style` is **applied** (rather than **set** using the `Style` property), only the differences between the new `Style` and existing `Style` properties will take effect. To completely replace a `Condition`'s `Style` with a new one, set Condition.Style.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueGreaterThan,
               "=100");
          cond.Style.BackgroundColor = Palette.SystemColor.Green;
          Style styleBold = wb.CreateStyle();
          styleBold.Font.Bold = true;
          cond.ApplyStyle(styleBold);
          Range rng = wb.Worksheets[0].CreateRange("$B$2:$F$10");
          rng.SetConditionalFormat(condFmt);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueGreaterThan, _
               "=100")
          cond.Style.BackgroundColor = Palette.SystemColor.Green
          Dim styleBold As Style = wb.CreateStyle()
          styleBold.Font.Bold = True
          cond.ApplyStyle(styleBold)
          Dim rng As Range = wb.Worksheets[0].CreateRange("$B$2:$F$10")
          rng.SetConditionalFormat(condFmt)
```

Condition.ChangeCondition

Overload	Description
ChangeCondition(Condition.Comparison, String)	Resets the current Condition's Comparison type and formula. The method takes one formulas, and therefore must take a `Comparison` that requires one formulas, such as Comparison.CellValueLessThan.
ChangeCondition(Condition.Comparison, String, String)	Resets the current Condition's Comparison type and minimum and maximum formulas. The method takes two formulas, and therefore must take a `Comparison` that requires two formulas, such as Comparison.CellValueBetween.

Condition.ChangeCondition(Condition.Comparison, String, String)

Description

Resets the current Condition's Comparison type and minimum and maximum formulas. The method takes two formulas, and therefore must take a `Comparison` that requires two formulas, such as Comparison.CellValueBetween.

Signature: C#

```
public void ChangeCondition(Comparison comparisonType, System.String formula1, System.String formula2)
```

Signature: VB.NET

```
Public Sub ChangeCondition(ByVal comparisonType As Comparison, ByVal formula1 As String, ByVal
formula2 As String)
```

Parameters

comparisonType

The type of comparison Excel will use when comparing a cell value to `formula1` and `formula2`. The comparison type must require two formulas and may be set to one of the following values: `Comparison.CellValueBetween` or Comparison.CellValueNotBetween.

formula1

The formula to use for the minumum value.

formula2

The formula to use for the maximum value.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueGreaterThan,
               "=100");
          cond.ChangeCondition(
               Condition.Comparison.CellValueBetween,
               "=100",
               "=150");
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueGreaterThan, _
               "=100")
          cond.ChangeCondition( _
               Condition.Comparison.CellValueBetween, _
               "=100", _
               "=150")
```

Condition.ChangeCondition(Condition.Comparison, String)

Description

Resets the current Condition's Comparison type and formula. The method takes one formulas, and therefore must take a `Comparison` that requires one formulas, such as Comparison.CellValueLessThan.

Signature: C#

```
public void ChangeCondition(Comparison comparisonType, System.String formula)
```

Signature: VB.NET

```
Public Sub ChangeCondition(ByVal comparisonType As Comparison, ByVal formula As String)
```

Parameters

comparisonType

The type of comparison Excel will use when comparing a cell value to the result of `formula`. The comparison type must require only one formula and may be set to one of the following values: Comparison.FormulaEvaluates, `Comparison.CellValueLessThan`, Comparison.CellValueGreaterThan, Comparison.CellValueLessThanOrEqualTo, Comparison.CellValueGreaterThanOrEqualTo, Comparison.CellValueEqualTo, or Comparison.CellValueNotEqualTo.

formula

The result of this formula will be compared to each cell value in the set of cells associated with the conditional format.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueBetween,
               "=100",
               "=150");
          cond.ChangeCondition(
               Condition.Comparison.CellValueGreaterThan, "=100");
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueBetween, _
               "=100", _
               "=150")
          cond.ChangeCondition( _
               Condition.Comparison.CellValueGreaterThan, "=100")
```

Condition.Comparison

Description

The value of Condition.Comparison determines whether the value of a conditionally formatted cell will be compared with one or two other values, or if a formula will be evaluated for each cell.

Signature: C#

```
public sealed class Comparison
```

Signature: VB.NET

```
Public NotInheritable Class Comparison
```

Values

Name	Description
FormulaEvaluation	The condition's Formula1 property will be a formula that evaluates to True or False.
CellValueBetween	Conditional formatting will be applied if a cell's value is between two specified values.
CellValueEqualTo	Conditional formatting will be applied if a cell's value is equal to the specified value.
CellValueGreaterThan	Conditional formatting will be applied if a cell's value is greater than the specified value.
CellValueGreaterThanOrEqualTo	Conditional formatting will be applied if a cell's value is greater than or equal to the specified value.

CellValueLessThan	Conditional formatting will be applied if a cell's value is less than the specified value.
CellValueLessThanOrEqualTo	Conditional formatting will be applied if a cell's value is less than or equal to the specified value.
CellValueNotBetween	Conditional formatting will be applied if a cell's value is not between two specified values.
CellValueNotEqualTo	Conditional formatting will be applied if a cell's value is not equal to the specified value.

Condition.ComparisonType

Description

Returns the Condition's Condition.Comparison value, which determines whether each cell value will be compared with one or two other values, or if a formula will be evaluated for each cell.

Signature: C#

```
public Comparison ComparisonType{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ComparisonType() As Comparison
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueGreaterThan,
               "=100");
          Condition.Comparison compType = cond.ComparisonType;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueGreaterThan, _
               "=100")
          Dim compType As Condition.Comparison = cond.ComparisonType
```

Condition.Formula1

Description

If the comparison is between a cell value and two values, this is the minimum value; otherwise this is is the formula to be evaluated for the condition.

Signature: C#

```
public System.String Formula1{ get; set; }
```

Signature: VB.NET

```
Public Property Formula1() As String
```

Remarks

If the value of Condition.ComparisonType is a comparison between a cell value and a single value (e.g., Condition.Comparison.CellValueEqualTo), Formula1 sets or returns a formula whose result will be compared with each cell value in the set of conditionally formatted cells.

If `Condition.ComparisonType` is a comparison between a cell value and two values (e.g., Condition.Comparison.CellValueBetween), `Formula1` sets or returns a formula whose result will be the minimum comparison value.

If the value of `Condition.ComparisonType` is Condition.Comparison.FormulaEvaluation, `Formula1` sets or returns a formula that evaluates to `true` or `false`.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueGreaterThan,
               "=100");
          cond.Formula1 = "=99";
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueGreaterThan, _
               "=100")
          cond.Formula1 = "=99"
```

Condition.Formula2

Description

If the value of Condition.ComparisonType is a comparison between a cell value and two values (e.g., Condition.Comparison.CellValueBetween), Formula2
sets or returns a formula whose result will be the maximum comparison value.

Signature: C#

```
public System.String Formula2{ get; set; }
```

Signature: VB.NET

```
Public Property Formula2() As String
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueBetween,
               "=100",
               "=150");
          cond.Formula2 = "=200";
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueBetween, _
               "=100", _
               "=150")
          cond.Formula2 = "=200"
```

Condition.Style

Description

Sets or returns the Style that will be set on conditionally formatted cells when the current Condition is met.

Signature: C#

```
public Style Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As Style
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Range rng = wb.Worksheets[0].CreateRange("$B$2:$F$10");
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueGreaterThan,
               "=100");
          Font fnt = cond.Style.Font;
          fnt.Bold = true;
          fnt.Color = wb.Palette.GetClosestColor(100, 100, 255);
          rng.SetConditionalFormat(condFmt);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim rng As Range = wb.Worksheets(0).CreateRange("$B$2:$F$10")
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueGreaterThan, _
               "=100")
          Dim fnt As Font = cond.Style.Font
          fnt.Bold = true fnt.Color = wb.Palette.GetClosestColor(100, 100, 255)
          rng.SetConditionalFormat(condFmt)
```

ConditionalFormat

Description

A ConditionalFormat object represents a conditional format that Excel will apply to a cell or group of cells if a specified condition is met. Each `ConditionalFormat` object contains up to three Condition and a Range object representing the set of cells on which the conditions will be tested.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class ConditionalFormat
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class ConditionalFormat
```

Remarks

To create a `ConditionalFormat`, call the Workbook object's CreateConditionalFormat method.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
Workbook wb = xla.Create();
ConditionalFormat cf = wb.CreateConditionalFormat();
```

vb.net

```
Dim xla As New ExcelApplication()
Dim wb As Workbook = xla.Create()
Dim cf As ConditionalFormat = wb.CreateConditionalFormat()
```

Properties

Name	Description
Count	Returns the number of Condition objects in the current ConditionalFormat object.
Range	Returns the Range of cells that is conditionally formatted by the current ConditionalFormat object.

Indexers

Name	Description
Item(Int32)	Returns the Condition object at the specified 0-based index.

Methods

Name	Description
CreateCondition(Comparison, String)	Creates a Condition of the specified ComparisonType.
CreateCondition(Comparison, String, String)	Creates a Condition of the specified ComparisonType.
RemoveCondition(Int32)	Removes the Condition at the specified index from the conditional format.
RemoveCondition(Condition)	Removes the specified Condition from the ConditionalFormat object.

ConditionalFormat.Count

Description

Returns the number of Condition objects in the current ConditionalFormat object.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat cf = wb.CreateConditionalFormat();
          int numConditions = cf.Count;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim cf As ConditionalFormat = wb.CreateConditionalFormat()
          Dim numConditions As Integer = cf.Count
```

ConditionalFormat.CreateCondition

Overload	Description
CreateCondition(Comparison, String)	Creates a Condition of the specified ComparisonType.
CreateCondition(Comparison, String, String)	Creates a Condition of the specified ComparisonType.

ConditionalFormat.CreateCondition(Condition.Comparison, String, String)

Description

Creates a Condition of the specified ComparisonType.

Signature: C#

```
public Condition CreateCondition(Comparison comparisonType, System.String formula1, System.String formula2)
```

Signature: VB.NET

```
Public Function CreateCondition(ByVal comparisonType As Comparison, ByVal formula1 As String, ByVal formula2 As String) As Condition
```

Parameters

comparisonType

A Condition constant representing the type of comparison Excel will use when comparing a cell value to the parameters `formula1` and `formula2`. The comparison type must require two formulas and may be set to one of the following values: Condition.Comparison.CellValueBetween or Condition.Comparison.CellValueNotBetween..

formula1

The formula to use for the minumum value.

formula2

The formula to use for the maximum value.

Returns

A Condition object representing the condition created.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Range rng = wb.Worksheest[0].CreateRange("$B$2:$F$10");
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueBetween,
               "=100",
               "=150");
          Font fnt = cond.Style.Font;
          fnt.Bold = true;
          fnt.Color = wb.Palette.GetClosestColor(100, 100, 255);
          rng.SetConditionalFormat(condFmt);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim rng As Range = wb.Worksheets(0).CreateRange("$B$2:$F$10")
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueBetween, _
               "=100", _
               "=150")
          Dim fnt As Font = cond.Style.Font
          fnt.Bold = True
          fnt.Color = wb.Palette.GetClosestColor(100, 100, 255)
          rng.SetConditionalFormat(condFmt)
```

ConditionalFormat.CreateCondition(Condition.Comparison, String)

Description

Creates a Condition of the specified ComparisonType.

Signature: C#

```
public Condition CreateCondition(Comparison comparisonType, System.String formula)
```

Signature: VB.NET

```
Public Function CreateCondition(ByVal comparisonType As Comparison, ByVal formula As String) As
Condition
```

Parameters

comparisonType

A Condition constant representing the type of comparison Excel will use when comparing a cell value to the result of the parameter `formula`. The comparison type must require only one formula and may be set to one of the following values: Condition.Comparison.FormulaEvaluates, Condition.Comparison.CellValueLessThan, Condition.Comparison.CellValueGreaterThan, Condition.Comparison.CellValueLessThanOrEqualTo, Condition.Comparison.CellValueGreaterThanOrEqualTo, Condition.Comparison.CellValueEqualTo, or Condition.Comparison.CellValueNotEqualTo.

formula

The result of this formula will be compared to each cell value in the set of cells associated with the conditional format.

Returns

A Condition object representing the condition created.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Range rng = wb.Worksheets[0].CreateRange("$B$2:$F$10");
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueGreaterThan,
               "=100");
          Font fnt = cond.Style.Font;
          fnt.Bold = true;
          fnt.Color = wb.Palette.GetClosestColor(100, 100, 255);
          rng.SetConditionalFormat(condFmt);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim rng As Range = wb.Worksheets(0).CreateRange("$B$2:$F$10")
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueGreaterThan, _
               "=100")
          Dim fnt As Font = cond.Style.Font
          fnt.Bold = True
          fnt.Color = wb.Palette.GetClosestColor(100, 100, 255)
          rng.SetConditionalFormat(condFmt)
```

ConditionalFormat.Item(Int32)

Description

Returns the Condition object at the specified 0-based index.

Signature: C#

```
public Condition this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As Condition
```

Parameters

index

The 0-based index of the `Condition` object.

Remarks

The index determines the order in which conditions will be evaluated. A ConditionalFormat may contain up to three `Condition` objects.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();

          //--- Create a range of the cells that will have the conditional formatting.
          Range vRange = wb.Worksheets[0].CreateRange("$B$2:$F$10");
          Area vArea = vRange.Areas[0];

          //--- Get the conditional format for the first cell within this range.
          ConditionalFormat vCndFmt = vArea[0,0].ConditionalFormat;

          //--- Modify the first condition.
          Condition vCondition = vCondFmt.Condition[0];
          vCondition.ChangeConditionType(Condition.Comparison.CellValueLessThan, "=100");

          vRange.SetConditionalFormat(vCondFmt);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()

          '--- Create a range of the cells that will have the conditional formatting.
          Dim rng As Range = wb.Worksheets(0).CreateRange("$B$2:$F$10")
          Dim ar As Area = rng.Areas(0)

          '--- Get the conditional format for the first cell within this range.
          Dim CondFmt As ConditionalFormat = ar(0,0).ConditionalFormat

          '--- Modify the first condition.
          Dim cond As Condition = CondFmt.Condition(0)
          cond.ChangeConditionType(Condition.Comparison.CellValueLessThan, "=100")

          rng.SetConditionalFormat(vCndFmt)
```

ConditionalFormat.Range

Description

Returns the Range of cells that is conditionally formatted by the current ConditionalFormat object.

Signature: C#

```
public Range Range{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Range() As Range
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat cf = wb.CreateConditionalFormat();
          Range cfRange = cf.Range;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim cf As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cfRange As Range = cf.Range
```

ConditionalFormat.RemoveCondition

Overload	Description
RemoveCondition(Condition)	Removes the specified Condition from the ConditionalFormat object.
RemoveCondition(Int32)	Removes the Condition at the specified index from the conditional format.

ConditionalFormat.RemoveCondition(Condition)

Description

Removes the specified Condition from the ConditionalFormat object.

Signature: C#

```
public void RemoveCondition(Condition condition)
```

Signature: VB.NET

```
Public Sub RemoveCondition(ByVal condition As Condition)
```

Parameters

condition

The Condition object to remove.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond1 =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueBetween,
               "=100",
               "=150");
          Condition cond2 =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueGreaterThan,
               "=100");
          condFmt.RemoveCondition(cond2);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond1 As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueGreaterThan, _
               "=100")
          Dim cond2 As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueBetween, _
               "=100", _
               "=150")
          condFmt.RemoveCondition(cond2)
```

ConditionalFormat.RemoveCondition(Int32)

Description

Removes the Condition at the specified index from the conditional format.

Signature: C#

```
public void RemoveCondition(int index)
```

Signature: VB.NET

```
Public Sub RemoveCondition(ByVal index As Integer)
```

Parameters

index

The 0-based index of the Condition object to remove.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ConditionalFormat condFmt = wb.CreateConditionalFormat();
          Condition cond1 =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueBetween,
               "=100",
               "=150");
          Condition cond2 =
               condFmt.CreateCondition(
               Condition.Comparison.CellValueGreaterThan,
               "=100");
          condFmt.RemoveCondition(0);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim  condFmt As ConditionalFormat = wb.CreateConditionalFormat()
          Dim cond1 As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueGreaterThan, _
               "=100")
          Dim cond2 As Condition = _
               condFmt.CreateCondition( _
               Condition.Comparison.CellValueBetween, _
               "=100", _
               "=150")
          condFmt.RemoveCondition(0)
```

CopyPasteProperties

Description

The CopyPasteProperties object represents the types of data that will be copied from a worksheet when you call the CopyPaste() method.

Signature: C#

```
public sealed class CopyPasteProperties
```

Signature: VB.NET

```
Public NotInheritable Class CopyPasteProperties
```

Remarks

You can use the CopyPasteType enumeration to specify the type of data that will be copied. If you set `CopyPasteType` to None, then you must set individual properties on the `CopyPasteProperties` object before calling the `CopyPaste()` method.

To create a CopyPasteProperties object, call the CreateCopyPasteProperties() method on a Workbook object.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          CopyPasteProperties properties =
               wb.CreateCopyPasteProperties(CopyPasteProperties.CopyPasteType.All);
          Area areaToCopy = wb.Worksheets[0].CreateArea(0, 0, 3, 3);
          Worksheet ws = wb.Worksheets[0];
          ws.CopyPaste(0, 0, areaToCopy, properties);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim properties As CopyPasteProperties = _
               wb.CreateCopyPasteProperties(CopyPasteProperties.CopyPasteType.All)
          Dim areaToCopy as Area = wb.Worksheets(0).CreateArea(0, 0, 3, 3)
          Dim ws as Worksheet = wb.Worksheets(0)
          ws.CopyPaste(0, 0, areaToCopy, properties)
```

Properties

Name	Description
CopyColumnWidth	A boolean value indicating whether or not to copy column width.
CopyComments	A boolean value indicating whether or not to copy comments. If this value is set to true, then any comments attached to cells in the copy area will be copied.
CopyFormatting	A boolean value indicating whether or not to copy cell formatting. Cell formatting includes font, color, number formatting, fill color, fill pattern and borders.
CopyFormulas	A boolean value indicating whether or not to copy cell formulas.
CopyMergedCells	A boolean value indicating whether or not to copy merged cells.
CopyRowHeight	A boolean value indicating whether or not to copy row height.
CopyValues	A boolean value indicating whether or not to copy cell values. If the CopyFormulas property is set to false, then the final value of any formulas will be copied.

Nested Classes

Name	Description
CopyPasteType	A CopyPasteType value specifies the type of data that will be copied.

CopyPasteProperties.CopyColumnWidth

Description

A boolean value indicating whether or not to copy column width.

Signature: C#

```
public boolean CopyColumnWidth{ get; set; }
```

Signature: VB.NET

```
Public Property CopyColumnWidth() As Boolean
```

Remarks

If this value is set to true, then column width of rows in the destination worksheet will be adjusted to match the column width from the corresponding column in the source worksheet. It is recommended that you do not turn on this setting without testing the results as multiple copy-paste operations into a worksheet may have undesired affects.

Examples

C#

```
//--- Is the property object set to copy column width?
          boolean willCopyColumnWidth = copyPasteProperties.CopyColumnWidth;

          //--- Set the property object to copy cell values.
          copyPasteProperties.CopyColumnWidth = true;
```

vb.net

```
'--- Is the property object set to copy column width?
          Dim willCopyColumnWidth As Boolean = copyPasteProperties.CopyColumnWidth

          '--- Set the property object to copy column width.
          copyPasteProperties.CopyColumnWidth = True
```

CopyPasteProperties.CopyComments

Description

A boolean value indicating whether or not to copy comments. If this value is set to true, then any comments attached to cells in the copy area will be copied.

Signature: C#

```
public boolean CopyComments{ get; set; }
```

Signature: VB.NET

```
Public Property CopyComments() As Boolean
```

Examples

C#

```
//--- Is the property object set to copy comments?
          boolean willCopyComments = copyPasteProperties.CopyComments;

          //--- Set the property object to copy cell values.
          copyPasteProperties.CopyComments = true;
```

vb.net

```
'--- Is the property object set to copy comments?
          Dim willCopyComments As Boolean = copyPasteProperties.CopyComments

          '--- Set the property object to copy comments.
          copyPasteProperties.CopyComments = True
```

CopyPasteProperties.CopyFormatting

Description

A boolean value indicating whether or not to copy cell formatting. Cell formatting includes font, color, number formatting, fill color, fill pattern and borders.

Signature: C#

```
public boolean CopyFormatting{ get; set; }
```

Signature: VB.NET

```
Public Property CopyFormatting() As Boolean
```

Examples

C#

```
//--- Is the property object set to copy cell formatting?
          boolean willCopyFormatting = copyPasteProperties.CopyFormatting;

          //--- Set the property object to copy cell formatting.
          copyPasteProperties.CopyFormatting = true;
```

vb.net

```
'--- Is the property object set to copy cell formatting?
          Dim willCopyFormatting As Boolean = copyPasteProperties.CopyFormatting

          '--- Set the property object to copy cell formatting.
          copyPasteProperties.CopyFormatting = True
```

CopyPasteProperties.CopyFormulas

Description

A boolean value indicating whether or not to copy cell formulas.

Signature: C#

```
public boolean CopyFormulas{ get; set; }
```

Signature: VB.NET

```
Public Property CopyFormulas() As Boolean
```

Remarks

If this property is set to true, then the formula will be copied. If this value is set to false and the CopyValues property is set to true, then only the final result of the formula, and not the formula itself, will be copied.

Examples

C#

```
//--- Is the property object set to copy formulas?
          boolean willCopyFormulas = copyPasteProperties.CopyFormulas;

          //--- Set the property object to copy formulas.
          copyPasteProperties.CopyFormulas = true;
```

vb.net

```
'--- Is the property object set to copy formulas?
          Dim willCopyFormulas As Boolean = copyPasteProperties.CopyFormulas

          '--- Set the property object to copy cell values.
          copyPasteProperties.CopyFormulas = True
```

CopyPasteProperties.CopyMergedCells

Description

A boolean value indicating whether or not to copy merged cells.

Signature: C#

```
public boolean CopyMergedCells{ get; set; }
```

Signature: VB.NET

```
Public Property CopyMergedCells() As Boolean
```

Remarks

If this value is set to false, any content inside merged cells in the source worksheet will be ignored.

Examples

C#

```
//--- Is the property object set to copy merged cells?
          boolean willCopyMergedCells = copyPasteProperties.CopyMergedCells;

          //--- Set the property object to copy cell values.
          copyPasteProperties.CopyMergedCells = true;
```

vb.net

```
'--- Is the property object set to copy merged cells?
          Dim willCopyMergedCells As Boolean = copyPasteProperties.CopyMergedCells

          '--- Set the property object to copy merged cells.
          copyPasteProperties.CopyMergedCells = True
```

CopyPasteProperties.CopyPasteType

Description

A CopyPasteType value specifies the type of data that will be copied.

Signature: C#

```
public sealed class CopyPasteType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class CopyPasteType
  Inherits TypesafeEnum
```

Values

Name	Description
All	Copies all currently supported types, except for row height and column width. Note: The behavior of `All` may change in the future as additional features are added to the CopyPaste method.
AllPlusRowHeightAndColumnWidth	Copies all currently supported types, including row height and column width. Note: If you call CopyPaste multiple times on the same worksheet, using this setting may have undesired results.
FormulasAndFormatting	Copy formulas and cell and number formatting.
None	Copies nothing. You must then set the behavior of CopyPaste by setting individual properties on the CopyPasteProperties object before calling the CopyPaste method.
ValuesAndFormatting	Copy cell values and cell and number formatting.
ValuesFormulasAndFormatting	Copy cell values, formulas, and cell and number formatting.
ValuesOnly	Copy only cell values.

CopyPasteProperties.CopyRowHeight

Description

A boolean value indicating whether or not to copy row height.

Signature: C#

```
public boolean CopyRowHeight{ get; set; }
```

Signature: VB.NET

```
Public Property CopyRowHeight() As Boolean
```

Remarks

If this value is set to true, then row height of rows in the destination worksheet will be adjusted to match the row height from the corresponding row in the source worksheet. It is recommended that you do not turn on this setting without testing the results as multiple copy-paste operations into a worksheet may have undesired affects.

Examples

C#

```
//--- Is the property object set to copy row height?
          boolean willCopyRowHeight = copyPasteProperties.CopyRowHeight;

          //--- Set the property object to copy cell values.
          copyPasteProperties.CopyRowHeight = true;
```

vb.net

```
'--- Is the property object set to copy row height?
          Dim willCopyRowHeight As Boolean = copyPasteProperties.CopyRowHeight

          '--- Set the property object to copy row height.
          copyPasteProperties.CopyRowHeight = True
```

CopyPasteProperties.CopyValues

Description

A boolean value indicating whether or not to copy cell values. If the CopyFormulas property is set to false, then the final value of any formulas will

be copied.

Signature: C#

```
public boolean CopyValues{ get; set; }
```

Signature: VB.NET

```
Public Property CopyValues() As Boolean
```

Examples

C#

```
//--- Is the property object set to copy cell values?
          boolean willCopyValues = copyPasteProperties.CopyValues;

          //--- Set the property object to copy cell values.
          copyPasteProperties.CopyValues = true;
```

vb.net

```
'--- Is the property object set to copy cell values?
          Dim willCopyValues As Boolean = copyPasteProperties.CopyValues

          '--- Set the property object to copy cell values.
          copyPasteProperties.CopyValues = True
```

DataImportProperties

Description

The DataImportProperties contains a set of properties that can be used when importing data into a Worksheet or Area. To return a `DataImportProperties` object, call Workbook.CreateDataImportProperties.

Signature: C#

```
public sealed class DataImportProperties
```

Signature: VB.NET

```
Public NotInheritable Class DataImportProperties
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          DataImportProperties importProps = wb.CreateDataImportProperties();
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim importProps As DataImportProperties = wb.CreateDataImportProperties()
```

Fields

Name	Description
ALLCOLUMNS	255
ALLROWS	65536

Properties

Name	Description
ColumnIndexFilter	Use ColumnIndexFilter to select specific columns within the imported data. Only the specified columns will be imported to the worksheet.
ColumnNameFilter	Use ColumnNameFilter to select specific columns within the imported data. Only the specified columns will be imported to the worksheet.
IgnoreNulls	Sets or returns whether NULL values in the imported dataset be ignored or converted to blank cells. Blank cells will override existing formulas or cell values.
MaxColumns	Sets or returns the maximum number of columns to import from the data source.
MaxRows	Sets or returns the maximum number of rows to import from the data source.
Transpose	When data is imported from a two-dimensional or rectangular array, the first dimension of the array corresponds to row and the second to column.
Truncate	Sets whether ExcelWriter should truncate imported data rows and columns that do not fit within the target rows and columns in the spreadsheet. If Truncate is not enabled, an exception will be thrown if the number of rows or columns imported exceeds the number of target rows or columns in the spreadsheet.
TruncateColumns	Sets or returns whether imported data columns will be truncated if the number of columns imported exceeds the number of target columns in the spreadsheet.
TruncateRows	Sets or returns whether imported data rows will be truncated if the number of rows imported exceeds the number of target rows in the spreadsheet.
UseColumnNames	Sets or returns whether to import column names to the first row (or first column if Transpose is set to `true`) of the imported data's target cells.

DataImportProperties.ALLCOLUMNS

Description

255

Signature: C#

```
public static const int ALLCOLUMNS = 256;
```

Signature: VB.NET

```
Public Shared Const ALLCOLUMNS As Integer = 256
```

DataImportProperties.ALLROWS

Description

65536

Signature: C#

```
public static const int ALLROWS = 65537;
```

Signature: VB.NET

```
Public Shared Const ALLROWS As Integer = 65537
```

DataImportProperties.ColumnIndexFilter

Description

Use ColumnIndexFilter to select specific columns within the imported data. Only the specified columns will be imported to the worksheet.

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 and Truncate is set to `true`, the smaller value will be used.

Using `ColumnIndexFilter` overwrites any previous ColumnNameFilter setting.

Examples

C#

```
//--- Get ColumnIndexFilter
          int[] arrFilter = importProps.ColumnIndexFilter;

          //--- Set ColumnIndexFilter
          importProps.ColumnIndexFilter = arrFilter;
```

vb.net

```
'--- Get ColumnIndexFilter
          Dim arrFilter() As Integer = importProps.ColumnIndexFilter

          '--- Set ColumnIndexFilter
          importProps.ColumnIndexFilter = arrFilter
```

DataImportProperties.ColumnNameFilter

Description

Use ColumnNameFilter to select specific columns within the imported data. Only the specified columns will be imported to the worksheet.

Signature: C#

```
public System.String[] ColumnNameFilter{ get; set; }
```

Signature: VB.NET

```
Public Property ColumnNameFilter() As String()
```

Remarks

If the length of the specified array is greater than MaxColumns, or exceeds the size of the target area and Truncate is set to `true`, the smaller

value will be used.

Using `ColumnNameFilter` overwrites any previous ColumnIndexFilter setting.

Examples

C#

```
//--- Get ColumnNameFilter
          string[] arrNames = importProps.ColumnNameFilter;

          //--- Set ColumnNameFilter
          importProps.ColumnNameFilter = arrNames;
```

vb.net

```
'--- Get ColumnNameFilter
          Dim arrNames() As String = importProps.ColumnNameFilter

          '--- Set ColumnNameFilter
          importProps.ColumnNameFilter = arrNames
```

DataImportProperties.IgnoreNulls

Description

Sets or returns whether NULL values in the imported dataset be ignored or converted to blank cells. Blank cells will override existing formulas or cell values.

Signature: C#

```
public boolean IgnoreNulls{ get; set; }
```

Signature: VB.NET

```
Public Property IgnoreNulls() As Boolean
```

Examples

C#

```
//--- Get IgnoreNulls
          bool ignoreNulls = importProps.ignoreNulls;

          //--- Set IgnoreNull
          importProps.IgnoreNulls = true;
```

vb.net

```
'--- Get IgnoreNulls
          Dim ignoreNulls As Boolean = importProps.IgnoreNulls

          '--- Set IgnoreNulls
          importProps.IgnoreNulls = True
```

DataImportProperties.MaxColumns

Description

Sets or returns the maximum number of columns to import from the data source.

Signature: C#

```
public int MaxColumns{ get; set; }
```

Signature: VB.NET

```
Public Property MaxColumns() As Integer
```

Examples

C#

```
//--- Get MaxColumns
          int maxCols = importProps.MaxColumns;

          //--- Set MaxColumns
          importProps.MaxColumns = 20;
```

vb.net

```
'--- Get MaxColumns
          Dim maxCols As Integer = importProps.MaxColumns

          '--- Set MaxColumns
          importProps.MaxColumns = 20
```

DataImportProperties.MaxRows

Description

Sets or returns the maximum number of rows to import from the data source.

Signature: C#

```
public int MaxRows{ get; set; }
```

Signature: VB.NET

```
Public Property MaxRows() As Integer
```

Examples

C#

```
//--- Get MaxRows
          int maxRows = importProps.MaxRows;

          //--- Set MaxRows
          importProps.MaxRows = 20;
```

vb.net

```
'--- Get MaxRows
          Dim maxRows As Integer = importProps.MaxRows

          '--- Set MaxRows
          importProps.MaxRows = 20
```

DataImportProperties.Transpose

Description

When data is imported from a two-dimensional or rectangular array, the first dimension of the array corresponds to row and the second to column.

Signature: C#

```
public boolean Transpose{ get; set; }
```

Signature: VB.NET

```
Public Property Transpose() As Boolean
```

Remarks

```
{{"A","X"},{"B","Y"},{"C","Z"}}
```

Would be entered in the worksheet as:

A	X
B	Y
C	Z

If Transpose is set to `true`, the format will be [column][row] and

```
{{"A","B","C"},{"X","Y","Z"}}
```

would be inserted as:

A	X
B	Y
C	Z

Transpose is a read/write property.

Examples

C#

```
//--- Get Transpose
          boolean transpose = importProps.Transpose;

          //--- Set Transpose
          importProps.Transpose = true;
```

vb.net

```
'--- Get Transpose
          Dim transpose As Boolean = importProps.Transpose

          '--- Set Transpose
          importProps.Transpose = True
```

DataImportProperties.Truncate

Description

Sets whether ExcelWriter should truncate imported data rows and columns that do not fit within the target rows and columns in the spreadsheet. If Truncate is not enabled, an exception will be thrown if the number of rows or columns imported exceeds the number of target rows or columns in the spreadsheet.

Signature: C#

```
public boolean Truncate{ set; }
```

Signature: VB.NET

```
Public WriteOnly Property Truncate() As Boolean
```

Examples

C#

```
//--- Get Truncate
          bool truncate = importProps.Truncate;

          //--- Set Truncate
          importProps.Truncate = true;
```

vb.net

```
'--- Get Truncate
          Dim truncate As Boolean = importProps.Truncate

          '--- Set Truncate
          importProps.Truncate = True
```

DataImportProperties.TruncateColumns

Description

Sets or returns whether imported data columns will be truncated if the number of columns imported exceeds the number of target columns in the spreadsheet.

Signature: C#

```
public boolean TruncateColumns{ get; set; }
```

Signature: VB.NET

```
Public Property TruncateColumns() As Boolean
```

Examples

C#

```
//--- Get TruncateColumns
          bool truncateCols = importProps.TruncateColumns;

          //--- Set TruncateColumns
          importProps.TruncateColumns = true;
```

vb.net

```
'--- Get TruncateColumns
          Dim truncateCols As Boolean = importProps.TruncateColumns

          '--- Set TruncateColumns
          importProps.TruncateColumns = True
```

DataImportProperties.TruncateRows

Description

Sets or returns whether imported data rows will be truncated if the number of rows imported exceeds the number of target rows in the spreadsheet.

Signature: C#

```
public boolean TruncateRows{ get; set; }
```

Signature: VB.NET

```
Public Property TruncateRows() As Boolean
```

Examples

C#

```
//--- Get TruncateRows
          bool truncateRows = importProps.TruncateRows;

          //--- Set TruncateRows
          importProps.TruncateRows = true;
```

vb.net

```
'--- Get TruncateRows
          Dim truncateRows As Boolean = importProps.TruncateRows

          '--- Set TruncateRows
          importProps.TruncateRows = True
```

DataImportProperties.UseColumnNames

Description

Sets or returns whether to import column names to the first row (or first column if Transpose is set to `true`) of the imported data's target cells.

Signature: C#

```
public boolean UseColumnNames{ get; set; }
```

Signature: VB.NET

```
Public Property UseColumnNames() As Boolean
```

Examples

C#

```
//--- Get UseColumnNames
          bool importColNames = importProps.UseColumnNames

          //--- Set UseColumnNames
          importProps.UseColumnNames = true;
```

vb.net

```
'--- Get UseColumnNames
          Dim importColNames As Boolean = oDataImportProperties.UseColumnNames

          '--- Set UseColumnNames
          importProps.UseColumnNames = True
```

DataLabelPlacementType

Description

The possible positions of a data label relative to its data point.

Signature: C#

```
public enum DataLabelPlacementType
```

Signature: VB.NET

```
Public Enum DataLabelPlacementType
```

Remarks

If you attempt to use a DataLabelPlacementType that does not correspond with the series' ChartType, you will receive the error "Error setting Data Label Placement: The data label placement type is not valid for this type of chart".

Examples

C#

```
//--- Create chart
Chart chart = ws.Charts.CreateChart(ChartType.Column.Stacked, ws.CreateAnchor(10, 0, 0, 0));

//--- Create data series
Series series = chart.SeriesCollection.CreateSeries("A1:A10");

//--- Set the data label position for the first SeriesDataLabel to OutsideEnd
series[0].Label.DataLabelPlacement = DataLabelPlacementType.OutsideEnd;
```

vb.net

```
'--- Create chart
Dim chart As Chart = ws.Charts.CreateChart(ChartType.Column.Stacked, ws.CreateAnchor(10, 0, 0, 0))

'--- Create data series
Dim series As Series = chart.SeriesCollection.CreateSeries("A1:A10")

'--- Set the data label position for the first SeriesDataLabel to OutsideEnd
series(0).Label.DataLabelPlacement = DataLabelPlacementType.OutsideEnd
```

Values

Name	Description
Default	The default data label position for column, bar, bubble, doughnut, line, and scatter charts
OutsideEnd	Places the data label on the outside of the data point in pie, column, and bar charts
InsideEnd	Places the data label inside the data point, near its edge in column, bar, and pie charts
Center	Places the data label in the center of the data point in bubble, line, and scatter charts
InsideBase	Places the data label on the inside of the data point, near its base in column and bar charts
Above	Places the data label above the data point in bubble, line, and scatter charts
Below	Places the data label below the data point in bubble, line, and scatter charts
Left	Places the data label to the left of the data point in bubble, line, and scatter charts
Right	Places the data label to the right of the data point in bubble, line, and scatter charts
Auto	Automatically place the data label depending on the space available. This is only available for pie charts, for which it sets the data label placement to "Best Fit"
MovedByUser	Place the data label at an arbitrary x and y position for any chart type

DataPoint

Description

A DataPoint object represents a data point in a chart. To return a `DataPoint` object, use Series[i].

Signature: C#

```
public sealed class DataPoint
```

Signature: VB.NET

```
Public NotInheritable Class DataPoint
```

Remarks

The `DataPoint` class extends ChartRegion.

Properties

Name	Description
DistanceFromCenter	Sets or returns the distance from the center of a pie chart as a percentage of the diameter.
Interior	Returns a ChartInterior object, representing the area of a data point.

IsOnSecondaryPieorBar	Sets or returns whether this data point is located on the secondary pie or bar in a Pie of Pie or Bar of Pie chart. For this to take effect, you must set SettingsPieDoughnut.BopPopSplit to Custom.
Label	Returns the SeriesDataLabel object for the data point.
LegendEntryFont	Sets or returns the font for this point's legend entry. This only applies when the series contains a single data point, as in pie charts.
Line	Returns a ChartLine object representing either the border around this data point - for most chart types - or the data point line itself, for line charts.
Marker	Returns a DataPointMarker object, representing the data marker of a point on a line, radar, or scatter series.
ShowLegendEntry	Sets or returns whether the entry in the legend for this series will be shown. This only applies when the series contains a single data point, as in pie charts.

DataPoint.DistanceFromCenter

Description

Sets or returns the distance from the center of a pie chart as a percentage of the diameter.

Signature: C#

```
public int DistanceFromCenter{ get; set; }
```

Signature: VB.NET

```
Public Property DistanceFromCenter() As Integer
```

DataPoint.Interior

Description

Returns a ChartInterior object, representing the area of a data point.

Signature: C#

```
public ChartInterior Interior{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Interior() As ChartInterior
```

DataPoint.IsOnSecondaryPieorBar

Description

Sets or returns whether this data point is located on the secondary pie or bar in a Pie of Pie or Bar of Pie chart. For this to take effect, you must set SettingsPieDoughnut.BopPopSplit to Custom.

Signature: C#

```
public boolean IsOnSecondaryPieorBar{ get; set; }
```

Signature: VB.NET

```
Public Property IsOnSecondaryPieorBar() As Boolean
```

DataPoint.Label

Description

Returns the SeriesDataLabel object for the data point.

Signature: C#

```
public SeriesDataLabel Label{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Label() As SeriesDataLabel
```

DataPoint.LegendEntryFont

Description

Sets or returns the font for this point's legend entry. This only applies when the series contains a single data point, as in pie charts.

Signature: C#

```
public Font LegendEntryFont{ get; set; }
```

Signature: VB.NET

```
Public Property LegendEntryFont() As Font
```

DataPoint.Line

Description

Returns a ChartLine object representing either the border around this data point - for most chart types - or the data point line itself, for line charts.

Signature: C#

```
public ChartLine Line{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Line() As ChartLine
```

DataPoint.Marker

Description

Returns a DataPointMarker object, representing the data marker of a point on a line, radar, or scatter series.

Signature: C#

```
public DataPointMarker Marker{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Marker() As DataPointMarker
```

DataPoint.ShowLegendEntry

Description

Sets or returns whether the entry in the legend for this series will be shown. This only applies when the series contains a single data point, as in pie charts.

Signature: C#

```
public boolean ShowLegendEntry{ get; set; }
```

Signature: VB.NET

```
Public Property ShowLegendEntry() As Boolean
```

DataPointMarker

Description

The DataPointMarker class represents the style and formatting of data markers in line, radar, and scatter charts. To return a `DataPointMarker` object, use Series.DataPointMarker.

Signature: C#

```
public sealed class DataPointMarker
```

Signature: VB.NET

```
Public NotInheritable Class DataPointMarker
```

Properties

Name	Description
BackgroundColor	Sets or returns the background color of series data markers.
ForegroundColor	Sets or returns the foreground color of series data markers.
HasShadow	Sets or returns whether data markers will be displayed with a shadow effect.
MarkerType	Sets or returns a data marker type for the series.
ShowBackgroundColor	Sets or returns whether the background color of series data markers will be visible or transparent.
ShowForegroundColor	Sets or returns whether the foreground color of series data markers will be visible or transparent.
Size	Sets or returns the size of series data markers, in points (1/72 inch).
Visible	Sets or returns whether data markers will be hidden or displayed.

Nested Classes

Name	Description
ShapeType	The ShapeType class contains all available data point marker shapes.

DataPointMarker.BackgroundColor

Description

Sets or returns the background color of series data markers.

Signature: C#

```
public Color BackgroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property BackgroundColor() As Color
```

DataPointMarker.ForegroundColor

Description

Sets or returns the foreground color of series data markers.

Signature: C#

```
public Color ForegroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property ForegroundColor() As Color
```

DataPointMarker.HasShadow

Description

Sets or returns whether data markers will be displayed with a shadow effect.

Signature: C#

```
public boolean HasShadow{ get; set; }
```

Signature: VB.NET

```
Public Property HasShadow() As Boolean
```

DataPointMarker.MarkerType

Description

Sets or returns a data marker type for the series.

Signature: C#

```
public ShapeType MarkerType{ get; set; }
```

Signature: VB.NET

```
Public Property MarkerType() As ShapeType
```

DataPointMarker.ShapeType

Description

The ShapeType class contains all available data point marker shapes.

Signature: C#

```
public sealed class ShapeType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ShapeType
  Inherits TypesafeEnum
```

Values

Name	Description
Circle	Circle
Diamond	Diamond
DowJones	Dow Jones
None	Data point markers will not be displayed.
PlusSign	Plus sign
Square	Square
StandardDeviation	Standard deviation
Star	Star
Triangle	Triangular
X	X

DataPointMarker.ShowBackgroundColor

Description

Sets or returns whether the background color of series data markers will be visible or transparent.

Signature: C#

```
public boolean ShowBackgroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property ShowBackgroundColor() As Boolean
```

DataPointMarker.ShowForegroundColor

Description

Sets or returns whether the foreground color of series data markers will be visible or transparent.

Signature: C#

```
public boolean ShowForegroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property ShowForegroundColor() As Boolean
```

DataPointMarker.Size

Description

Sets or returns the size of series data markers, in points (1/72 inch).

Signature: C#

```
public int Size{ get; set; }
```

Signature: VB.NET

```
Public Property Size() As Integer
```

DataPointMarker.Visible

Description

Sets or returns whether data markers will be hidden or displayed.

Signature: C#

```
public boolean Visible{ get; set; }
```

Signature: VB.NET

```
Public Property Visible() As Boolean
```

DataValidation

Description

A DataValidation object represents a rule for validating data entered in cells. The rule may be applied to a cell or a set of cells. The Workbook class contains several CreateDataValidation methods for creating `DataValidation` objects. To apply a `DataValidation` object to a cell or group of cells, use Area.SetDataValidation or Range.SetDataValidation.

Signature: C#

```
public sealed class DataValidation
```

Signature: VB.NET

```
Public NotInheritable Class DataValidation
```

Remarks

In Microsoft Excel, data validation rules are created through the **Data Validation** dialog. To open this dialog, open the **Data** menu and select **Validation**.

Examples

C#

```
ExcelApplication xlw = new ExcelApplication();
          Workbook wb = xlw.Create();

          //--- Create a data validation rule:
          DataValidation dv = wb.CreateDataValidation(
               DataValidation.ValidationType.WholeNumber,
               DataValidation.ComparisonType.Between,
               "=1", "=100");
          String formula = "Sheet1!A2:F33 Sheet2!A5:D15";

          //--- Apply the rule to a Range:
          wb.CreateRange(formula).SetDataValidation(dv);
```

vb.net

```
Dim xlw As New ExcelApplication()
          Dim wb As Workbook = xlw.Create()

          '--- Create a data validation rule:
          Dim dv As DataValidation = wb.CreateDataValidation( _
               DataValidation.ValidationType.WholeNumber, _
               DataValidation.ComparisonType.Between, _
               "=1", "=100")
          Dim formula As String = "Sheet1!A2:F33 Sheet2!A5:D15"

          '--- Apply the rule to a Range:
          wb.CreateRange(formula).SetDataValidation(dv)
```

Properties

Name	Description
AllowedType	Returns the type of value that may be entered in a cell.
Comparison	Returns the type of comparison that will be used to validate a value entered in a cell with a data validation rule.
ErrorAlert	Sets or returns the content of the error alert message displayed when invalid data is entered in a cell.
ErrorAlertStyle	Sets or returns the style of error alert to display if error alerts are enabled.
ErrorAlertTitle	Sets or returns the title of the error alert dialog that is displayed when invalid data is entered in a cell.
IgnoreBlanks	Sets or returns whether the data validation rule should ignore blank cells.
InputMessage	Sets or returns the content of the input message dialog that is displayed when the user selects a cell with a data validation rule.
MinimumValue	Sets or returns the minimum value that may be entered in a cell with a data validation rule.
MaximumValue	Sets or returns the maximum value that may be entered in a cell with a data validation rule.
ShowErrorAlert	Sets or returns whether Excel should display an error alert if invalid data is entered.
ShowInputMessage	Sets or returns whether to show a message when the user selects a cell to which a data validation rule applies.
Title	Sets or returns the title of the input message dialog that is displayed when the user selects a cell with a data validation rule.
UseListDropdown	Sets or returns whether a drop-down list will be used to display valid values.

Methods

Name	Description
Clear()	Sets all properties of the DataValidation object to an uninitialized state.
SetAllowedType(ValidationType, ComparisonType, Object)	Sets the allowed data type, comparison type, and minimum value to use when validating a cell entry.
SetAllowedType(ValidationType, ComparisonType, Object, Object)	Sets the allowed data type, comparison type, and minimum and maximum values to use when validating a cell entry.
SetAllowedType(ValidationType, Object)	Sets the allowed data type and a set of valid values to use when validating a cell entry.

Nested Classes

Name	Description
ComparisonType	The type of comparison that will be used for validating the value of the cell.
ErrorAlertStyleType	Type style of the icon in the error alert message box.
ValidationType	The data type allowed as the cell's value.

DataValidation.AllowedType

Description

Returns the type of value that may be entered in a cell.

Signature: C#

```
public ValidationType AllowedType{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property AllowedType() As ValidationType
```

DataValidation.Clear()

Description

Sets all properties of the DataValidation object to an uninitialized state.

Signature: C#

```
public void Clear()
```

Signature: VB.NET

```
Public Sub Clear()
```

Examples

C#

```
cell.DataValidation.Clear();
```

vb.net

```
cell.DataValidation.Clear()
```

DataValidation.Comparison

Description

Returns the type of comparison that will be used to validate a value entered in a cell with a data validation rule.

Signature: C#

```
public ComparisonType Comparison{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Comparison() As ComparisonType
```

Remarks

For validation types where ComparisonType is not allowed, (List, Custom, and AnyValue) this method will throw an exception.

DataValidation.ComparisonType

Description

The type of comparison that will be used for validating the value of the cell.

Signature: C#

```
public sealed class ComparisonType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ComparisonType
  Inherits TypesafeEnum
```

Values

Name	Description
Between	Between
EqualTo	Equal to
GreaterThan	Greater than
GreaterThanOrEqualTo	Greater than or equal to
LessThan	Less than
LessThanOrEqualTo	Less than or equal to
NotBetween	Not between
NotEqualTo	Not Equal to

DataValidation.ErrorAlert

Description

Sets or returns the content of the error alert message displayed when invalid data is entered in a cell.

Signature: C#

```
public System.String ErrorAlert{ get; set; }
```

Signature: VB.NET

```
Public Property ErrorAlert() As String
```

DataValidation.ErrorAlertStyle

Description

Sets or returns the style of error alert to display if error alerts are enabled.

Signature: C#

```
public ErrorAlertStyleType ErrorAlertStyle{ get; set; }
```

Signature: VB.NET

```
Public Property ErrorAlertStyle() As ErrorAlertStyleType
```

DataValidation.ErrorAlertStyleType

Description

Type style of the icon in the error alert message box.

Signature: C#

```
public sealed class ErrorAlertStyleType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ErrorAlertStyleType
  Inherits TypesafeEnum
```

Values

Name	Description
Information	A white bubble with a blue 'i'. The message box buttons will be: 'OK' and 'Cancel'.
Stop	A red circle with a white 'X'. The message box buttons will be: 'Retry' and 'Cancel'.
Warning	A yellow triangle with a black '!'. The message box buttons will be: 'Yes', 'No', and 'Cancel'.

DataValidation.ErrorAlertTitle

Description

Sets or returns the title of the error alert dialog that is displayed when invalid data is entered in a cell.

Signature: C#

```
public System.String ErrorAlertTitle{ get; set; }
```

Signature: VB.NET

```
Public Property ErrorAlertTitle() As String
```

DataValidation.IgnoreBlanks

Description

Sets or returns whether the data validation rule should ignore blank cells.

Signature: C#

```
public boolean IgnoreBlanks{ get; set; }
```

Signature: VB.NET

```
Public Property IgnoreBlanks() As Boolean
```

DataValidation.InputMessage

Description

Sets or returns the content of the input message dialog that is displayed when the user selects a cell with a data validation rule.

Signature: C#

```
public System.String InputMessage{ get; set; }
```

Signature: VB.NET

```
Public Property InputMessage() As String
```

DataValidation.MaximumValue

Description

Sets or returns the maximum value that may be entered in a cell with a data validation rule.

Signature: C#

```
public System.Object MaximumValue{ get; set; }
```

Signature: VB.NET

```
Public Property MaximumValue() As Object
```

Remarks

See object documentation for the ValidationType value type combinations accepted by this method. For ValidationTypes (AnyValue, List, Custom) and ComparisonTypes (anything other than Between and NotBetween) which do not support a maximum value, this method will throw an exception.

DataValidation.MinimumValue

Description

Sets or returns the minimum value that may be entered in a cell with a data validation rule.

Signature: C#

```
public System.Object MinimumValue{ get; set; }
```

Signature: VB.NET

```
Public Property MinimumValue() As Object
```

Remarks

See object documentation for the ValidationType value type combinations accepted by this method. For ValidationTypes which do not support a minimum value, (DataValidation.ValidationType.AnyValue) this method will throw an exception.

DataValidation.SetAllowedType

Overload	Description
SetAllowedType(ValidationType, ComparisonType, Object)	Sets the allowed data type, comparison type, and minimum value to use when validating a cell entry.
SetAllowedType(ValidationType, ComparisonType, Object, Object)	Sets the allowed data type, comparison type, and minimum and maximum values to use when validating a cell entry.
SetAllowedType(ValidationType, Object)	Sets the allowed data type and a set of valid values to use when validating a cell entry.

DataValidation.SetAllowedType(DataValidation.ValidationType, DataValidation.ComparisonType, Object, Object)

Description

Sets the allowed data type, comparison type, and minimum and maximum values to use when validating a cell entry.

Signature: C#

```
public void SetAllowedType(ValidationType allowedType, ComparisonType comparison, System.Object
minimumValue, System.Object maximumValue)
```

Signature: VB.NET

```
Public Sub SetAllowedType(ByVal allowedType As ValidationType, ByVal comparison As ComparisonType,
ByVal minimumValue As Object, ByVal maximumValue As Object)
```

Parameters

allowedType

The data type allowed in the cell.

comparison

The comparison used to validate the cell's value.

minimumValue

A primitive or Date representing the cell's maximum value.

maximumValue

A primitive or Date representing the cell's minimum value.

DataValidation.SetAllowedType(DataValidation.ValidationType, DataValidation.ComparisonType, Object)

Description

Sets the allowed data type, comparison type, and minimum value to use when validating a cell entry.

Signature: C#

```
public void SetAllowedType(ValidationType allowedType, ComparisonType comparison, System.Object value)
```

Signature: VB.NET

```
Public Sub SetAllowedType(ByVal allowedType As ValidationType, ByVal comparison As ComparisonType,
ByVal value As Object)
```

Parameters

allowedType

The data type allowed in the cell.

comparison

The comparison used to validate the cell's value.

value

A primitive or Date representing data validation's constraint value.

Remarks

This method may only be used for ValidationTypes other than List and Custom. This method may only be used for ComparisonTypes other than Between and NotBetween. See object documentation for the ValidationType parameter type combinations accepted by this method.

DataValidation.SetAllowedType(DataValidation.ValidationType, Object)

Description

Sets the allowed data type and a set of valid values to use when validating a cell entry.

Signature: C#

```
public void SetAllowedType(ValidationType allowedType, System.Object sourceArray)
```

Signature: VB.NET

```
Public Sub SetAllowedType(ByVal allowedType As ValidationType, ByVal sourceArray As Object)
```

Parameters

allowedType

The data type allowed in the cell.

sourceArray

For 'List' allowed type: the array of valid values for the cell.

Remarks

This method may only be used for ValidationTypes of List, Custom, or AnyValue. See object documentation for the ValidationType parameter type combinations accepted by this method.

DataValidation.ShowErrorAlert

Description

Sets or returns whether Excel should display an error alert if invalid data is entered.

Signature: C#

```
public boolean ShowErrorAlert{ get; set; }
```

Signature: VB.NET

```
Public Property ShowErrorAlert() As Boolean
```

DataValidation.ShowInputMessage

Description

Sets or returns whether to show a message when the user selects a cell to which a data validation rule applies.

Signature: C#

```
public boolean ShowInputMessage{ get; set; }
```

Signature: VB.NET

```
Public Property ShowInputMessage() As Boolean
```

DataValidation.Title

Description

Sets or returns the title of the input message dialog that is displayed when the user selects a cell with a data validation rule.

Signature: C#

```
public System.String Title{ get; set; }
```

Signature: VB.NET

```
Public Property Title() As String
```

DataValidation.UseListDropdown

Description

Sets or returns whether a drop-down list will be used to display valid values.

Signature: C#

```
public boolean UseListDropdown{ get; set; }
```

Signature: VB.NET

```
Public Property UseListDropdown() As Boolean
```

DataValidation.ValidationType

Description

The data type allowed as the cell's value.

Signature: C#

```
public sealed class ValidationType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ValidationType
  Inherits TypesafeEnum
```

Values

Name	Description
AnyValue	Anything
Custom	A custom type
Date	A date
Decimal	A floating point number
List	A list
TextLength	A text list
Time	A time
WholeNumber	A whole number

DocumentProperties

Description

The DocumentProperties object represents properties of a spreadsheet. These properties are displayed in the file's Properties sheet which can be

opened in Excel (File -> Properties), or by right-clicking the file icon and selecting "Properties." The `DocumentProperties` class can be used to remove personal information from an existing spreadsheet, or to populate searchable fields that can be used to locate the file.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class DocumentProperties
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class DocumentProperties
```

Remarks

To return a `DocumentProperties` object, use Workbook.DocumentProperties.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          DocumentProperties docProps = wb.DocumentProperties;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim docProps = wb.DocumentProperties
```

Properties

Name	Description
Application	Sets or Returns the name of the Application that created the spreadsheet.
Author	Sets or returns the author of the spreadsheet. Excel allows you to search for a file by author.
Category	Sets or returns the category of the spreadsheet. Excel allows you to search for a file by category.
Comments	Sets or returns the value of the Comments field in a workbook's Properties sheet. Excel allows you to search for a file using this value.
Company	Sets or returns the value of the Company field in a workbook's Properties sheet. Excel allows you to search for a file using this value.
CreationDate	Sets or returns the workbook's creation date. By default, CreationDate returns the spreadsheet's actual creation date. You can set `CreationDate` to a different date, but this will not change the creation date displayed in the spreadsheet's Properties sheet.
Keywords	Sets or returns keywords that can be used to search for the spreadsheet. In Excel, a file's keywords are displayed in the Summary page of the Properties sheet.
LastSavedDate	Sets or returns the date on which the current spreadsheet was last saved. This value is not set by default, is not displayed in the spreadsheet's Properties sheet, and cannot be used to search for a file.
Manager	Sets or returns the value of the Manager field in the spreadsheet's Properties sheet. Excel allows you to search for a file using this value.
Title	Sets or returns the value of the Title field in a workbook's Properties sheet. Excel allows you to search for a file using this value.

Indexers

Name	Description
Item(DocumentProperties.BuiltIn)	Returns the value of a specified built-in property. The value can be one of the following types, depending on the property: String, Double, Integer, Boolean or DateTime.
Item(String)	Returns the value of a specified property. The value can be one of the following types, depending on the property: String, Double, Integer, Boolean or DateTime.

Methods

Name	Description
Clear()	Clears all properties contained in the DocumentProperties object, removing them from the workbook's Properties sheet.
GetCustomProperty(String)	Returns the value of the custom property for the specified key. The return value can be of the following types, depending on the property: String, Double, Integer, Boolean or DateTime. (To create a custom property, use DocumentProperties.SetCustomProperty.)
RestoreDefaults()	Restores document properties values to the default set of properties and clears all others. The default set of properties are Application, CreationDate, and LastSavedDate.
SetCustomProperty(String, Object)	Use SetCustomProperty to create a custom document property. Pass the method the name and value of the property. The value can be one of the following types, depending on the property: String, Double, Integer, Boolean, or DateTime.

Nested Classes

Name	Description
BuiltIn	DocumentProperties.BuiltIn contains a set of built-in document property fields.

DocumentProperties.Application

Description

Sets or Returns the name of the Application that created the spreadsheet.

Signature: C#

```
public System.String Application{ get; set; }
```

Signature: VB.NET

```
Public Property Application() As String
```

Examples

C#

```
//--- Get Application
          string app = docProps.Application;

          //--- Set Application
          docProps.Application = "SoftArtisans ExcelWriter";
```

vb.net

```
'--- Get Application
          Dim app As String = docProps.Application

          '--- Set Application
          docProps.Application = "SoftArtisans ExcelWriter"
```

DocumentProperties.Author

Description

Sets or returns the author of the spreadsheet. Excel allows you to search for a file by author.

Signature: C#

```
public System.String Author{ get; set; }
```

Signature: VB.NET

```
Public Property Author() As String
```

Examples

C#

```
//--- Get Application
          string author = docProps.Author;

          //--- Set Author
          docProps.Author = "J. Smith";
```

vb.net

```
'--- Get Application
          Dim author As String = docProps.Author

          '--- Set Author
          docProps.Author = "J. Smith"
```

DocumentProperties.BuiltIn

Description

DocumentProperties.BuiltIn contains a set of built-in document property fields.

Signature: C#

```
public sealed class BuiltIn : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BuiltIn
  Inherits TypesafeEnum
```

Values

Name	Description
Application	Application
Author	Author
Category	Category
Comments	Comments
Company	Company
CreationDate	File creation date

EditTime	Edit time
Keywords	Keywords
LastPrintedDate	Last date the workbook was printed
LastSavedBy	User that saved the workbook last
LastSavedDate	Date the workbook was last save
LinksUpToDate	Whether or not links in the workbook are up-to-date
Manager	Manager
Notes	Notes
RevisionNumber	Revision number
ScaleOrCrop	Scale or crop
Subject	Subject
Title	Title

DocumentProperties.Category

Description

Sets or returns the category of the spreadsheet. Excel allows you to search for a file by category.

Signature: C#

```
public System.String Category{ get; set; }
```

Signature: VB.NET

```
Public Property Category() As String
```

Examples

C#

```
//--- Get Category
          string category = docProps.Category;

          //--- Set Category
          docProps.Category = "Sales Reports";
```

vb.net

```
'--- Get Category
          Dim category As String = docProps.Category

          '--- Set Category
          docProps.Category = "Sales Reports"
```

DocumentProperties.Clear()

Description

Clears all properties contained in the DocumentProperties object, removing them from the workbook's Properties sheet.

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 the value of the Comments field in a workbook's Properties sheet. Excel allows you to search for a file using this value.

Signature: C#

```
public System.String Comments{ get; set; }
```

Signature: VB.NET

```
Public Property Comments() As String
```

Examples

C#

```
//--- Get Comments
          string comments = docProps.Comments;

          //--- Set Comments
          docProps.Comments = "Generated by ExcelWriter";
```

vb.net

```
'--- Get Comments
          Dim comments As String = docProps.Comments

          '--- Set Comments
          docProps.Comments = "Generated by ExcelWriter"
```

DocumentProperties.Company

Description

Sets or returns the value of the Company field in a workbook's Properties sheet. Excel allows you to search for a file using this value.

Signature: C#

```
public System.String Company{ get; set; }
```

Signature: VB.NET

```
Public Property Company() As String
```

Examples

C#

```
//--- Get Company
          string company = docProps.Company;

          //--- Set Company
          docProps.Company = "SoftArtisans";
```

vb.net

```
'--- Get Company
          Dim company As String = docProps.Company

          '--- Set Company
          docProps.Company = "SoftArtisans"
```

DocumentProperties.CreationDate

Description

Sets or returns the workbook's creation date. By default, CreationDate returns the spreadsheet's actual creation date. You can set `CreationDate` to a different date, but this will not change the creation date displayed in the spreadsheet's Properties sheet.

Signature: C#

```
public System.DateTime CreationDate{ get; set; }
```

Signature: VB.NET

```
Public Property CreationDate() As Date
```

Examples

C#

```
//--- Get CreationDate
          DateTime createDate = docProps.CreationDate;

          //--- Set CreationDate
          docProps.CreationDate = new DateTime(2005, 08, 04);
```

vb.net

```
'--- Get CreationDate
          Dim createDate As DateTime = docProps.CreationDate

          '--- Set CreationDate
          docProps.CreationDate = New DateTime(2005, 08, 04)
```

DocumentProperties.GetCustomProperty(String)

Description

Returns the value of the custom property for the specified key. The return value can be of the following types, depending on the property: String, Double, Integer, Boolean or DateTime. (To create a custom property, use DocumentProperties.SetCustomProperty.)

Signature: C#

```
public System.Object GetCustomProperty(System.String key)
```

Signature: VB.NET

```
Public Function GetCustomProperty(ByVal key As String) As Object
```

Parameters

key

The key or name of the custom property.

Returns

The value of the custom property.

Examples

C#

```
Object reviewedBy = docProps.GetCustomProperty("Reviewed by");
```

vb.net

```
Dim reviewedBy As Object = docProps.GetCustomProperty("Reviewed by")
```

DocumentProperties.Item

Overload	Description
Item(BuiltIn)	Returns the value of a specified built-in property. The value can be one of the following types, depending on the property: String, Double, Integer, Boolean or DateTime.
Item(String)	Returns the value of a specified property. The value can be one of the following types, depending on the property: String, Double, Integer, Boolean or DateTime.

DocumentProperties.Item(DocumentProperties.BuiltIn)

Description

Returns the value of a specified built-in property. The value can be one of the following types, depending on the property: String, Double, Integer, Boolean or DateTime.

Signature: C#

```
public System.Object this[BuiltIn property] { get; set; }
```

Signature: VB.NET

```
Public Default Property Item(ByVal [property] As BuiltIn) As Object
```

Parameters

property

The built-in property.

Returns

The value of the built-in property.

Examples

C#

```
//--- Get Property
          Object title = docProps[DocumentProperties.BuiltIn.Title];

          //--- Set Property
          docProps[DocumentProperties.BuiltIn.Title] = "xlw";
```

vb.net

```
'--- Get Property
          Dim title As Object = docProps(DocumentProperties.BuiltIn.Title)

          '--- Set Property
          docProps(DocumentProperties.BuiltIn.Title) = "xlw"
```

DocumentProperties.Item(String)

Description

Returns the value of a specified property. The value can be one of the following types, depending on the property: String, Double, Integer, Boolean or DateTime.

Signature: C#

```
public System.Object this[System.String propertyName] { get; set; }
```

Signature: VB.NET

```
Public Default Property Item(ByVal propertyName As String) As Object
```

Parameters

property

The built-in property.

Returns

The value of the built-in property.

Examples

C#

```
//--- Get Property
          Object application = docProps["Application"];

          //--- Set Property
          docProps["Application"] = "xlw";
```

vb.net

```
'--- Get Property
          Dim application As Object = docProps("Application")

          '--- Set Property
          docProps("Application") = "xlw"
```

DocumentProperties.Keywords

Description

Sets or returns keywords that can be used to search for the spreadsheet. In Excel, a file's keywords are displayed in the Summary page of the Properties sheet.

Signature: C#

```
public System.String Keywords{ get; set; }
```

Signature: VB.NET

```
Public Property Keywords() As String
```

Examples

C#

```
//--- Get Keywords
          string keywords = docProps.Keywords;

          //--- Set Keywords
          docProps.Keywords = "ExcelWriter, WordWriter, OfficeWriter";
```

vb.net

```
'--- Get Keywords
          Dim keywords As String = docProps.Keywords

          '--- Set Keywords
          docProps.Keywords = "ExcelWriter, WordWriter, OfficeWriter"
```

DocumentProperties.LastSavedDate

Description

Sets or returns the date on which the current spreadsheet was last saved. This value is not set by default, is not displayed in the spreadsheet's Properties sheet, and cannot be used to search for a file.

Signature: C#

```
public System.DateTime LastSavedDate{ get; set; }
```

Signature: VB.NET

```
Public Property LastSavedDate() As Date
```

Examples

C#

```
//--- Get LastSavedDate
          DateTime savedDate = docProps.LastSavedDate;

          //--- Set LastSavedDate
          docProps.LastSavedDate = new DateTime(2005, 08, 04);
```

vb.net

```
'--- Get LastSavedDate
          Dim savedDate As DateTime = docProps.LastSavedDate

          '--- Set LastSavedDate
          docProps.LastSavedDate = New DateTime(2005, 08, 04)
```

DocumentProperties.Manager

Description

Sets or returns the value of the Manager field in the spreadsheet's Properties sheet. Excel allows you to search for a file using this value.

Signature: C#

```
public System.String Manager{ get; set; }
```

Signature: VB.NET

```
Public Property Manager() As String
```

Examples

C#

```
//--- Get Manager
          string manager = docProps.Manager;

          //--- Set Manager
          docProps.Manager = "J. Smith";
```

vb.net

```
'--- Get Manager
          Dim manager As String = docProps.Manager

          '--- Set Manager
          docProps.Manager = "J. Smith"
```

DocumentProperties.RestoreDefaults()

Description

Restores document properties values to the default set of properties and clears all others. The default set of properties are Application, CreationDate, and LastSavedDate.

Signature: C#

```
public void RestoreDefaults()
```

Signature: VB.NET

```
Public Sub RestoreDefaults()
```

Examples

C#

```
docProps.RestoreDefaults();
```

vb.net

```
docProps.RestoreDefaults()
```

DocumentProperties.SetCustomProperty(String, Object)

Description

Use SetCustomProperty to create a custom document property. Pass the method the name and value of the property. The value can be one of the following types, depending on the property: String, Double, Integer, Boolean, or DateTime.

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

The key or name of the custom property.

value

The value of the custom property.

Examples

C#

```
docProps.SetCustomProperty("Reviewed by", "J. Smith");
```

vb.net

```
docProps.SetCustomProperty("Reviewed by", "J. Smith")
```

DocumentProperties.Title

Description

Sets or returns the value of the Title field in a workbook's Properties sheet. Excel allows you to search for a file using this value.

Signature: C#

```
public System.String Title{ get; set; }
```

Signature: VB.NET

```
Public Property Title() As String
```

Examples

C#

```
//--- Get Title
          string title = docProps.Title;

          //--- Set Title
          docProps.Title = "Employees";
```

vb.net

```
'--- Get Title
          Dim title As String = docProps.Title

          '--- Set Title
          docProps.Title = "Employees"
```

ErrorBar

Description

An ErrorBar object represents series error bars. Error bars graphically express potential error amounts relative to each data marker in a data series. You can add error bars to data series in two-dimensional area, bar, column, line, scatter, and bubble charts. In scatter and bubble charts, you can display error bars for x values, y values, or both.

Signature: C#

```
public sealed class ErrorBar
```

Signature: VB.NET

```
Public NotInheritable Class ErrorBar
```

Remarks

To return an `ErrorBar` object representing Y Error bars, use Series.ErrorBarX.

Properties

Name	Description
CustomMinusFormula	Sets or returns a custom negative error value for the error bars.
CustomPlusFormula	Sets or returns a custom positive error value for the error bars.
Direction	Sets or returns the direction in which the error bars will be displayed.
Line	Returns the ChartLine object used to format the appearance of the error bar's line.
TShaped	Sets or returns whether the error bars will be T shaped.
Value	Sets or returns the error amount. The meaning of this value is dependent on the value of ErrorBar.ErrorValueType.
ValueType	Sets or returns the type of value to use as the error amount. Depending on the error type, the amount may be a fixed value, percentage, standard deviation, standard error, or a custom value.

Nested Classes

Name	Description
BarDirection	The BarDirection class contains the available direction combinations for error bar display.
ErrorValueType	An ErrorValueType value specifies an error bar's error value type.

ErrorBar.BarDirection

Description

The BarDirection class contains the available direction combinations for error bar display.

Signature: C#

```
public sealed class BarDirection : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BarDirection
  Inherits TypesafeEnum
```

Values

Name	Description
Both	Both positive and negative error bars will be displayed.
Minus	Negative error bars will be displayed.
None	Error bars will not be displayed.
Plus	Positive error bars will be displayed.

ErrorBar.CustomMinusFormula

Description

Sets or returns a custom negative error value for the error bars.

Signature: C#

```
public System.String CustomMinusFormula{ get; set; }
```

Signature: VB.NET

```
Public Property CustomMinusFormula() As String
```

ErrorBar.CustomPlusFormula

Description

Sets or returns a custom positive error value for the error bars.

Signature: C#

```
public System.String CustomPlusFormula{ get; set; }
```

Signature: VB.NET

```
Public Property CustomPlusFormula() As String
```

ErrorBar.Direction

Description

Sets or returns the direction in which the error bars will be displayed.

Signature: C#

```
public BarDirection Direction{ get; set; }
```

Signature: VB.NET

```
Public Property Direction() As BarDirection
```

ErrorBar.ErrorValueType

Description

An ErrorValueType value specifies an error bar's error value type.

Signature: C#

```
public sealed class ErrorValueType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ErrorValueType
  Inherits TypesafeEnum
```

Values

Name	Description
Custom	The error value specified by ErrorBar.Value is a custom value.
FixedValue	The error value specified by ErrorBar.Value is a fixed value.
Percentage	The error value specified by ErrorBar.Value is a percentage.
StandardDeviation	The error value specified by ErrorBar.Value is a standard deviation.
StandardError	Standard error

ErrorBar.Line

Description

Returns the ChartLine object used to format the appearance of the error bar's line.

Signature: C#

```
public ChartLine Line{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Line() As ChartLine
```

ErrorBar.TShaped

Description

Sets or returns whether the error bars will be T shaped.

Signature: C#

```
public boolean TShaped{ get; set; }
```

Signature: VB.NET

```
Public Property TShaped() As Boolean
```

ErrorBar.Value

Description

Sets or returns the error amount. The meaning of this value is dependent on the value of ErrorBar.ErrorValueType.

Signature: C#

```
public double Value{ get; set; }
```

Signature: VB.NET

```
Public Property Value() As Double
```

ErrorBar.ValueType

Description

Sets or returns the type of value to use as the error amount. Depending on the error type, the amount may be a fixed value, percentage, standard deviation, standard error, or a custom value.

Signature: C#

```
public ErrorValueType ValueType{ get; set; }
```

Signature: VB.NET

```
Public Property ValueType() As ErrorValueType
```

ExcelApplication

Description

ExcelApplication is the main class for pure code-based workbook generation (for information about template-based workbook generation, see ExcelTemplate). This class is an engine used to open, create, and write (save or stream to a browser) workbooks. A single instance of `ExcelApplication` can generate multiple workbooks.

Signature: C#

```
public class ExcelApplication
```

Signature: VB.NET

```
Public Class ExcelApplication
```

Remarks

A generated workbook is represented by the Workbook class.

To create an instance of `ExcelApplication`:

In C#:

- Import `ExcelApplication` to the C# page:

```
using SoftArtisans.OfficeWriter.ExcelWriter;
```

- Create an `ExcelApplication` object:

```
ExcelApplication xla = new ExcelApplication();
```

In VB.NET:

- Import `ExcelApplication` to the VB.NET page:

```
Imports SoftArtisans.OfficeWriter.ExcelWriter
```

- Create an `ExcelApplication` object:

```
Dim xla As New ExcelApplication()
```

Properties

Name	Description
LicenseKey	The LicenseKey property is provided to assist in debugging license-related issues. The property sets or returns the license key that enables ExcelWriter. The key is stored in the registry. It contains information about ExcelWriter, including product name, platform, and version.
Version	Returns the exact version of ExcelWriter. The format of the version string returned is:

Methods

Name	Description
Create()	Creates a new workbook containing 1 worksheet. The sheet is blank when the workbook is created, and standard font, style, and format settings applied.
ExcelApplication()	Creates a new ExcelApplication object
Open(ExcelTemplate)	Opens a Workbook from an ExcelTemplate object.
Open(String)	Opens an existing Excel spreadsheet.
Open(String, String)	Opens an existing Excel spreadsheet.
Open(System.IO.Stream)	Opens an Excel workbook from a stream and returns a Workbook object.
Open(System.IO.Stream, String)	Opens an Excel workbook from a stream and returns a Workbook object.
Save(Workbook, String)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.
Save(Workbook, System.IO.Stream)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later).xls file.
Save(Workbook, System.Web.HttpResponse, String, Boolean)	Streams the specified workbook to the client. It should be the only method that uses the response object, as it automatically sets response headers, clears the body of the response, and flushes it to the client.

Save(Workbook, System.Web.HttpResponse, String, Boolean, String)	Streams the specified workbook to the client. It should be the only method that uses the response object, as it automatically sets response headers, clears the body of the response, and flushes it to the client.

Extension Methods

Introduced in build 7.5

Overload	Description
Open(Microsoft.SharePoint.SPDocumentLibrary, String)	Opens an existing Excel spreadsheet from a SharePoint Document Library.
Open(Microsoft.SharePoint.SPListItem, String)	Opens an existing Excel spreadsheet that is attached to a SharePoint List Item.
Save(Workbook, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.
Save(Workbook, Microsoft.SharePoint.SPListItem, String)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.

ExcelApplication.Create()

Description

Creates a new workbook containing 1 worksheet. The sheet is blank when the workbook is created, and standard font, style, and format settings applied.

Signature: C#

```
public virtual Workbook Create()
```

Signature: VB.NET

```
Public Overridable Function Create() As Workbook
```

Returns

A Workbook object representing the created workbook.

Examples

C#

```
Workbook wb = xla.Create();
```

vb.net

```
Dim wb As Workbook = xla.Create()
```

ExcelApplication.ExcelApplication()

Description

Creates a new ExcelApplication object

Signature: C#

```
public ExcelApplication ( )
```

Signature: VB.NET

```
Public Sub New ExcelApplication ( )
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
```

vb.net

```
Dim xla As New ExcelApplication()
```

ExcelApplication.LicenseKey

Description

The LicenseKey property is provided to assist in debugging license-related issues. The property sets or returns the license key that enables ExcelWriter. The key is stored in the registry. It contains information about ExcelWriter, including product name, platform, and version.

Signature: C#

```
public System.String LicenseKey{ get; set; }
```

Signature: VB.NET

```
Public Property LicenseKey() As String
```

Remarks

For assistance with licensing issues, contact SoftArtisans Technical Services (http://support.softartisans.com\).

Examples

C#

```
//--- Get LicenseKey
          string version = xla.LicenseKey;

          //--- Set LicenseKey
          xla.LicenseKey = "XXXXXX-XXXX-XXXX-XXXXXX";
```

vb.net

```
'--- Get LicenseKey
          Dim version As String = xla.LicenseKey

          '--- Set LicenseKey
          xla.LicenseKey = "XXXXXX-XXXX-XXXX-XXXXXX"
```

ExcelApplication.Open

Overload	Description
Open(ExcelTemplate)	Opens a Workbook from an ExcelTemplate object.
Open(String)	Opens an existing Excel spreadsheet.
Open(String, String)	Opens an existing Excel spreadsheet.

Open(System.IO.Stream)	Opens an Excel workbook from a stream and returns a Workbook object.
Open(System.IO.Stream, String)	Opens an Excel workbook from a stream and returns a Workbook object.

SharePoint Integration Extension Methods

Overload	Description
Open(Microsoft.SharePoint.SPDocumentLibrary, String)	Opens an existing Excel spreadsheet from a SharePoint Document Library.
Open(Microsoft.SharePoint.SPListItem, String)	Opens an existing Excel spreadsheet that is attached to a SharePoint List Item.

ExcelApplication.Open(ExcelTemplate)

Description

Opens a Workbook from an ExcelTemplate object.

Signature: C#

```
public virtual Workbook Open(ExcelTemplate excelTemplate)
```

Signature: VB.NET

```
Public Overridable Function Open(ByVal excelTemplate As ExcelTemplate) As Workbook
```

Parameters

excelTemplate

An `ExcelTemplate` object representing the workbook to open.

Returns

Workbook that is opened.

Exceptions

System.Exception

If there is a problem creating, opening, or writing to the file specified, or reading from the `Workbook` object.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();

          //--- Do something with template here

          ExcelApplication xla = new ExcelApplication();
          xla.Open(xlt);
```

vb.net

```
Dim xlt As New ExcelTemplate()

          '--- Do something with template here

          Dim xla As New ExcelApplication()
          xla.Open(xlt)
```

ExcelApplication.Open(String, String)

Description

Opens an existing Excel spreadsheet.

Signature: C#

```
public virtual Workbook Open(System.String fileName, System.String decryptPassword)
```

Signature: VB.NET

```
Public Overridable Function Open(ByVal fileName As String, ByVal decryptPassword As String) As
Workbook
```

Parameters

fileName

The path and name of the file to open. The path can be relative or absolute. The file must be a BIFF8 format (Excel 97 or later) .xls file.

decryptPassword

The password used to encrypt the spreadsheet. Standard "Office 97/2000 Compatible" encryption is used to encrypt the spreadsheet. This is the default encryption used by Excel 97 and later versions.

Returns

A Workbook object representing the file opened.

Exceptions

System.Exception

If the file cannot be found or opened, or if the file is not a BIFF8 format (Excel 97 or later) .xls file.

Remarks

ExcelWriter supports Excel's BIFF8 (Excel 97 or later) format only. Do not use this method to open BIFF7 (Excel 95) files.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Open(@"C:\Reports\Report.xls", "MyPassword");
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Open("C:\Reports\Report.xls", "MyPassword")
```

ExcelApplication.Open(String)

Description

Opens an existing Excel spreadsheet.

Signature: C#

```
public virtual Workbook Open(System.String fileName)
```

Signature: VB.NET

```
Public Overridable Function Open(ByVal fileName As String) As Workbook
```

Parameters

fileName

The path and name of the file to open. The path can be relative or absolute. The file must be a BIFF8 format (Excel 97 or later) .xls file.

Returns

A Workbook object representing the file opened.

Exceptions

System.Exception

If the file cannot be found or opened, or if ExcelWriter cannot parse the file as a BIFF8 file.

Remarks

ExcelWriter supports Excel's BIFF8 (Excel 97 or later) format only. Do not use this method to open BIFF7 (Excel 95) files.

Examples

C#

```
Workbook wb = xla.Open(@"C:\Reports\Report.xls");
```

vb.net

```
Dim wb As Workbook = xla.Open("C:\Reports\Report.xls")
```

ExcelApplication.Open(System.IO.Stream, String)

Description

Opens an Excel workbook from a stream and returns a Workbook object.

Signature: C#

```
public virtual Workbook Open(System.IO.Stream stream, System.String decryptPassword)
```

Signature: VB.NET

```
Public Overridable Function Open(ByVal stream As System.IO.Stream, ByVal decryptPassword As String) As
Workbook
```

Parameters

stream

A System.IO.Stream containing a BIFF8 format (Excel 97 or later) .xls file.

decryptPassword

The password used to encrypt the spreadsheet. Standard "Office 97/2000 Compatible" encryption is used to encrypt the spreadsheet. This is the default encryption used by Excel 97 and later versions.

Returns

A `Workbook` object representing the file to open.

Exceptions

System.Exception

If the file cannot be found or opened, or if the file is not a BIFF8 format (Excel 97 or later) .xls file.

Remarks

ExcelWriter supports Excel's BIFF8 (Excel 97 or later) format only. Do not use this method to open BIFF7 (Excel 95) files.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          FileStream fStrm = new FileStream(@"C:\Sales\2003\June.xls", FileMode.Open);
          Workbook wb = xla.Open(fStrm, "MyPassword");
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim fStrm As FileStream = New FileStream("C:\Sales\2003\June.xls")
          Dim wb As Workbook = xla.Open(fStrm, "MyPassword")
```

ExcelApplication.Open(System.IO.Stream)

Description

Opens an Excel workbook from a stream and returns a Workbook object.

Signature: C#

```
public virtual Workbook Open(System.IO.Stream stream)
```

Signature: VB.NET

```
Public Overridable Function Open(ByVal stream As System.IO.Stream) As Workbook
```

Parameters

stream

A System.IO.Stream containing a BIFF8 format (Excel 97 or later) .xls file.

Returns

A `Workbook` object representing the file opened.

Exceptions

System.Exception

If the file cannot be found or opened, or if the file is not a BIFF8 format (Excel 97 or later) .xls file.

Remarks

ExcelWriter supports Excel's BIFF8 (Excel 97 or later) format only. Do not use this method to open BIFF7 (Excel 95) files.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          FileStream fStrm = new FileStream(@"C:\Sales\2003\June.xls", FileMode.Open);
          Workbook wb = xla.Open(fStrm);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim fStrm As FileStream = New FileStream("C:\Sales\2003\June.xls", FileMode.Open)
          Dim wb As Workbook = xla.Open(fStrm)
```

ExcelApplication.Save

Overload	Description
Save(Workbook, String)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.
Save(Workbook, System.IO.Stream)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later).xls file.
Save(Workbook, System.Web.HttpResponse, String, Boolean)	Streams the specified workbook to the client. It should be the only method that uses the response object, as it automatically sets response headers, clears the body of the response, and flushes it to the client.
Save(Workbook, System.Web.HttpResponse, String, Boolean, String)	Streams the specified workbook to the client. It should be the only method that uses the response object, as it automatically sets response headers, clears the body of the response, and flushes it to the client.

SharePoint Integration Extension Methods

Overload	Description
Save(Workbook, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.
Save(Workbook, Microsoft.SharePoint.SPListItem, String)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.

ExcelApplication.Save(Workbook, String)

Description

Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.

Signature: C#

```
public virtual void Save(Workbook workbook, System.String fileName)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal workbook As Workbook, ByVal fileName As String)
```

Parameters

workbook

A `Workbook` object representing the workbook to save.

fileName

Specifies a complete path and file name for the Workbook. ExcelWriter will save the file to this location. If a file with the same name exists, it will be overwritten by the new Excel file.

Exceptions

System.Exception

If there is a problem creating, opening, or writing to the file specified, or reading from the workbook object.

Examples

C#

```
ExcelApplication oExcelApp = new ExcelApplication();
          Workbook wb = oExcelApp.Create();
          oExcelApp.Save(wb,@"C:\Sales2003\June.xls");
```

vb.net

```
Dim oExcelApp As New ExcelApplication()
          Dim wb As Workbook = oExcelApp.Create()
          oExcelApp.Save(wb, "C:\Sales2003\June.xls")
```

ExcelApplication.Save(Workbook, System.IO.Stream)

Description

Saves a specified Workbook as a BIFF8 format (Excel 97 or later).xls file.

Signature: C#

```
public virtual void Save(Workbook workbook, System.IO.Stream stream)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal workbook As Workbook, ByVal stream As System.IO.Stream)
```

Parameters

workbook

A `Workbook` object representing the workbook to save.

stream

The output stream to write to. This stream must be open and writable. ExcelWriter will not close this stream; the calling program is responsible for closing the stream and releasing the resource.

Exceptions

System.Exception

If there is a problem creating, opening, or writing to the file specified, or reading from the workbook object.

Examples

C#

```
ExcelApplication oExcelApp = new ExcelApplication();
          Workbook wb = oExcelApp.Create();
          FileStream fstream = new FileStream(@"C:\temp\outfile.xls",
               FileMode.Create);
          oExcelApp.Save(wb,fstream);
```

vb.net

```
Dim oExcelApp As New ExcelApplication()
          Dim wb As Workbook = oExcelApp.Create()
          Dim fstream As New FileStream("C:\temp\outfile.xls", _
               FileMode.Create)
          oExcelApp.Save(wb, fstream)
```

ExcelApplication.Save(Workbook, System.Web.HttpResponse, String, Boolean, String)

Description

Streams the specified workbook to the client. It should be the only method that uses the response object, as it automatically sets response headers, clears the body of the response, and flushes it to the client.

Signature: C#

```
public virtual void Save(Workbook workbook, System.Web.HttpResponse response, System.String fileName,
boolean openInBrowser, System.String contentType)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal workbook As Workbook, ByVal response As System.Web.HttpResponse,
ByVal fileName As String, ByVal openInBrowser As Boolean, ByVal contentType As String)
```

Parameters

workbook

A Workbook object representing the workbook to save.

response

A `System.Http.Response` object, usually Page.Response.

fileName

Specifies a file name for the workbook. This name will be displayed in the download dialog when the file is streamed to the browser.

openInBrowser

If true, the spreadsheet will be opened in place in Internet Explorer, rather than spawning a seperate Excel window.

contentType

The content-type header sent to the browser with the generated spreadsheet.

Exceptions

System.Exception

If there is an error obtaining the response stream from the Response object.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = oExcelApp.Create();

          //--- Do something here
          xla.Save(wb, Page.Response, "generated.xls", false, "application/unknown");
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()

          '--- Do something here
          xla.Save(wb, Page.Response, "generated.xls", False, "application/unknown")
```

ExcelApplication.Save(Workbook, System.Web.HttpResponse, String, Boolean)

Description

Streams the specified workbook to the client. It should be the only method that uses the response object, as it automatically sets response headers, clears the body of the response, and flushes it to the client.

Signature: C#

```
public virtual void Save(Workbook workbook, System.Web.HttpResponse response, System.String fileName,
boolean openInBrowser)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal workbook As Workbook, ByVal response As System.Web.HttpResponse,
ByVal fileName As String, ByVal openInBrowser As Boolean)
```

Parameters

workbook

A Workbook object representing the workbook to save.

response

A `System.Http.Response` object, usually Page.Response.

fileName

Specifies a file name for the workbook. This name will be displayed in the download dialog when the file is streamed to the browser.

openInBrowser

If true, the spreadsheet will be opened in place in Internet Explorer, rather than spawning a seperate Excel window.

Exceptions

System.Exception

If there is an error obtaining the response stream from the Response object.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();

          //--- Do something here
          xla.Save(wb, Page.Response, "generated.xls", false);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()

          '--- Do something here
          xla.Save(wb, Page.Response, "generated.xls", False)
```

ExcelApplication.Version

Description

Returns the exact version of ExcelWriter. 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 = xla.Version;
```

vb.net

```
Dim ver As String = xla.Version
```

ExcelWriterException

Description

All methods in the ExcelWriter API can throw an exception of type ExcelWriterException.

Signature: C#

```
[Serializable]
 public sealed class ExcelWriterException : System.Exception,
System.Runtime.Serialization.ISerializable
```

Signature: VB.NET

```
<Serializable()> _
 Public NotInheritable Class ExcelWriterException
  Inherits System.Exception
  Implements System.Runtime.Serialization.ISerializable
```

Font

Description

The Font object represents a font style that you can use when creating a reusable Style object, or when setting the style of a cell or set of cells. To set or return a `Font` object, use Style.Font.

Signature: C#

```
public sealed class Font
```

Signature: VB.NET

```
Public NotInheritable Class Font
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          SoftArtisans.OfficeWriter.ExcelWriter.Style styl = wb.CreateStyle();
          SoftArtisans.OfficeWriter.ExcelWriter.Font fnt = styl.Font;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim styl As SoftArtisans.OfficeWriter.ExcelWriter.Style = wb.CreateStyle()
          Dim fnt As SoftArtisans.OfficeWriter.ExcelWriter.Font = styl.Font
```

Properties

Name	Description
Bold	Sets or returns whether text will be displayed in bold face.
Color	Sets or returns the color of the font.
Italic	Sets or returns whether text will be italicized.
Name	Sets or returns the name of the font (e.g., "Arial"). The font must be available on the client's system for Excel to use it; if it is not available, Excel will use a default font.
ScriptStyle	Sets or returns the vertical position of the font in relation to other text.
Size	Sets or returns the font size, in points.
Strikethrough	Sets or returns whether strikethrough formatting will be applied to the font.
Underline	Sets or returns the underline style for this font.

Nested Classes

Name	Description
UnderlineStyle	An UnderlineStyle value specifies an underline style for a font.
VerticalPosition	A vertical value specifies the vertical position of a font in relation to other text.

Font.Bold

Description

Sets or returns whether text will be displayed in bold face.

Signature: C#

```
public boolean Bold{ get; set; }
```

Signature: VB.NET

```
Public Property Bold() As Boolean
```

Examples

C#

```
//--- Get Bold
          bool isBold = fnt.Bold;

          //--- Set Bold
          fnt.Bold = true;
```

vb.net

```
'--- Get Bold
          Dim isBold As Boolean = fnt.Bold

          '--- Set Bold
          fnt.Bold = True
```

Font.Color

Description

Sets or returns the color of the font.

Signature: C#

```
public Color Color{ get; set; }
```

Signature: VB.NET

```
Public Property Color() As Color
```

Examples

C#

```
//--- Get Color
          Color fontClr = fnt.Color;

          //--- Set Color
          fnt.Color = wb.Palette.GetClosestColor(100, 100, 255);
```

vb.net

```
'--- Get Color
          Dim fontClr As Color = fnt.Color

          '--- Set Color
          fnt.Color = wb.Palette.GetClosestColor(100, 100, 255)
```

Font.Italic

Description

Sets or returns whether text will be italicized.

Signature: C#

```
public boolean Italic{ get; set; }
```

Signature: VB.NET

```
Public Property Italic() As Boolean
```

Examples

C#

```
//--- Get Italic
          bool isItalic = fnt.Italic;

          //--- Set Italic
          fnt.Italic = true;
```

vb.net

```
'--- Get Italic
          Dim isItalic As Boolean = fnt.Italic

          '--- Set Italic
          fnt.Italic = True
```

Font.Name

Description

Sets or returns the name of the font (e.g., "Arial"). The font must be available on the client's system for Excel to use it; if it is not available, Excel will use a default font.

Signature: C#

```
public System.String Name{ get; set; }
```

Signature: VB.NET

```
Public Property Name() As String
```

Examples

C#

```
//--- Get Name
          string fontName = fnt.Name;

          //--- Set Name
          fnt.Name = "Times New Roman";
```

vb.net

```
'--- Get Name
          Dim fontName As String = fnt.Name

          '--- Set Name
          fnt.name = "Times New Roman"
```

Font.ScriptStyle

Description

Sets or returns the vertical position of the font in relation to other text.

Signature: C#

```
public VerticalPosition ScriptStyle{ get; set; }
```

Signature: VB.NET

```
Public Property ScriptStyle() As VerticalPosition
```

Remarks

If the vertical position is Superscript, the font will be slightly higher than other text. If the vertical position is Subscript, the font will be slightly lower than other text. If the vertical position is set to None, the font will be displayed normally.

Examples

C#

```
//--- Get ScriptStyle
          Font.VerticalPosition scrStyle = fnt.ScripstStyle;

          //--- Set ScriptStyle
          fnt.ScriptStyle = Font.VerticalPosition.Subscript;
```

vb.net

```
'--- Get ScriptStyle
          Dim scrStyle As Font.VerticalPosition = fnt.ScriptStyle

          '--- Set ScriptStyle
          fnt.ScriptStyle = Font.VerticalPosition.Subscript
```

Font.Size

Description

Sets or returns the font size, in points.

Signature: C#

```
public double Size{ get; set; }
```

Signature: VB.NET

```
Public Property Size() As Double
```

Examples

C#

```
//--- Get Size
          double fontSize = fnt.Size;

          //--- Set Size
          fnt.Size = 12;
```

vb.net

```
'--- Get Size
          Dim fontSize As Double = fnt.Size

          '--- Set Size
          fnt.Size = 12
```

Font.Strikethrough

Description

Sets or returns whether strikethrough formatting will be applied to the font.

Signature: C#

```
public boolean Strikethrough{ get; set; }
```

Signature: VB.NET

```
Public Property Strikethrough() As Boolean
```

Examples

C#

```
//--- Get Strikethrough
          bool isStrikethrough = fnt.Strikethrough;

          //--- Set Strikethrough
          fnt.Strikethrough = true;
```

vb.net

```
'--- Get Strikethrough
          Dim isStrikethrough As Boolean = fnt.Strikethrough

          '--- Set Strikethrough
          fnt.Strikethrough = True
```

Font.Underline

Description

Sets or returns the underline style for this font.

Signature: C#

```
public UnderlineStyle Underline{ get; set; }
```

Signature: VB.NET

```
Public Property Underline() As UnderlineStyle
```

Examples

C#

```
//--- Get Underline
          Font.UnderlineStyle underStyle = fnt.Underline;

          //--- Set Underline
          fnt.Underline = Font.UnderlineStyle.Double;
```

vb.net

```
'--- Get Underline
          Dim underStyle As Font.UnderlineStyle = fnt.Underline

          '--- Set Underline
          fnt.Underline = Font.UnderlineStyle.Double
```

Font.UnderlineStyle

Description

An UnderlineStyle value specifies an underline style for a font.

Signature: C#

```
public sealed class UnderlineStyle : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class UnderlineStyle
  Inherits TypesafeEnum
```

Values

Name	Description
Accounting	Accounting style underline.
Double	Double underline.
DoubleAccounting	Double accounting underline.
None	Text is not underlined.
Single	Single underline.

Font.VerticalPosition

Description

A vertical value specifies the vertical position of a font in relation to other text.

Signature: C#

```
public sealed class VerticalPosition : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class VerticalPosition
  Inherits TypesafeEnum
```

Values

Name	Description
None	Standard vertical position
Subscript	The font will be slightly lower than other text.
Superscript	The font will be slightly higher than other text.

GlobalStyle

Description

ExcelWriter creates three style types: GlobalStyle, NamedStyle, and CellStyle. All three style types derive from the Style class, and `NamedStyle` derives from `GlobalStyle`.

Signature: C#

```
public class GlobalStyle : Style
```

Signature: VB.NET

```
Public Class GlobalStyle
  Inherits Style
```

Remarks

A style can be set or applied to cells, rows, columns, ranges, and areas. When a style is set, it is cloned and the object to which the style is assigned acquires all of that style's properties, including font properties and number formatting. When a style is applied, only the differences between the new style and existing style properties (assigned through the ExcelWriter API or in Microsoft Excel) will take effect. For example, if the cell has a background color and the new style applied does not contain a background color, the cell's color will not be affected. However, if the new style includes a background color, it will replace the existing background color of the cell.

To set a style, use the `Style` property. To apply a style, call `ApplyStyle`. Both are accessible through the following objects: Cell, Area, Range, RowProperties, and ColumnProperties.

Examples

C#

```
Style styleMoneyFormat = wb.CreateStyle();
          styleMoneyFormat.NumberFormat = "$#,##0";
          Style styleTitle = wb.CreateStyle();
          styleTitle.Font.Size = 14;
          styleTitle.Font.Color = wb.Palette.GetClosestColor(100, 100, 255);
```

vb.net

```
Dim styleMoneyFormat As Style = wb.CreateStyle()
          styleMoneyFormat.NumberFormat = "$#,##0"
          Dim styleTitle As Style = wb.CreateStyle()
          styleTitle.Font.Size = 14
          styleTitle.Font.Color = wb.Palette.GetClosestColor(100, 100, 255)
```

Group

Description

A Group is a type of shape that contains other shapes. Currently, groups may not be created or have their contents modified. They may be moved, resized, and deleted.

Signature: C#

```
public class Group
```

Signature: VB.NET

```
Public Class Group
```

Properties

Name	Description
Height	Sets or returns the height of the Group, in points (1/72 inch).
Position	Sets or returns the Group 's position in the worksheet.
Type	Returns the ShapeType of the group, which is always ShapeType.Group.
Width	Sets or returns the width of the Group, in points (1/72 inch).

Group.Height

Description

Sets or returns the height of the Group, in points (1/72 inch).

Signature: C#

```
public double Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Double
```

Group.Position

Description

Sets or returns the Group 's position in the worksheet.

Signature: C#

```
public Anchor Position{ get; set; }
```

Signature: VB.NET

```
Public Property Position() As Anchor
```

Group.Type

Description

Returns the ShapeType of the group, which is always ShapeType.Group.

Signature: C#

```
public ShapeType Type{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Type() As ShapeType
```

Group.Width

Description

Sets or returns the width of the Group, in points (1/72 inch).

Signature: C#

```
public double Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Double
```

Groups

Description

Groups provides access to the collection of Group objectss contained within a drawing. Currently, only top level groups may be accessed. Nested groups are not supported. Additionally, a group may not be created or have its contents modified. However, a group can be moved, resized, and removed from a drawing. To return a `Groups` collection, use Worksheet.ShapeGroups.

Signature: C#

```
public sealed class Groups : System.Collections.Generic.IEnumerable<Group>
```

Signature: VB.NET

```
Public NotInheritable Class Groups
  Implements System.Collections.Generic.IEnumerable(Of Group)
```

Properties

Name	Description
Count	Returns the number of Group objects in the collection.

Methods

Name	Description
GetEnumerator()	Returns an IEnumerator for the Groups in a Worksheet
ListGroups()	Returns an array of all the Group objects in the collection.
Remove(Group)	Removes the specified Group object from the collection.
Remove(Int32)	Removes the Group object at the specified 0-based position in the collection.

Groups.Count

Description

Returns the number of Group objects in the collection.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Groups.GetEnumerator()

Description

Returns an IEnumerator for the Groups in a Worksheet

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the Groups in a Worksheet as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = ws.ShapeGroups.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = ws.ShapeGroups.GetEnumerator()
```

Groups.ListGroups()

Description

Returns an array of all the Group objects in the collection.

Signature: C#

```
public Group[] ListGroups()
```

Signature: VB.NET

```
Public Function ListGroups() As Group()
```

Returns

An array of all the `Group` objects in the collection.

Groups.Remove

Overload	Description

Remove(Group)	Removes the specified Group object from the collection.
Remove(Int32)	Removes the Group object at the specified 0-based position in the collection.

Groups.Remove(Group)

Description

Removes the specified Group object from the collection.

Signature: C#

```
public void Remove(Group group)
```

Signature: VB.NET

```
Public Sub Remove(ByVal group As Group)
```

Parameters

group

The `Group` object to remove.

Groups.Remove(Int32)

Description

Removes the Group object at the specified 0-based position in the collection.

Signature: C#

```
public void Remove(int position)
```

Signature: VB.NET

```
Public Sub Remove(ByVal position As Integer)
```

Parameters

position

The 0-based index of the `Group` object to remove.

HeaderFooterPicture

Introduced in build 4.5.1

Description

A HeaderFooterPicture object represents a picture embedded in a Header or Footer.

Signature: C#

```
public sealed class HeaderFooterPicture
```

Signature: VB.NET

```
Public NotInheritable Class HeaderFooterPicture
```

Remarks

Use this class in order to set the size of a picture within a HeaderFooterSection. In order to return a HeaderFooterPicture object, use the HeaderFooterSection.GetPicture() method.

Examples

C#

```
HeaderFooterPicture hfp = header.GetPicture();

//--- Set height to 20 pixels
hfp.Height = 20;
```

vb.net

```
Dim hfp As HeaderFooterPicture = header.GetPicture()

'--- Set height to 20 pixels
hfp.Height = 20
```

Properties

Name	Description
Height	Sets the height in pixels of a picture embedded within a header or footer
LockAspectRatio	Locks the aspect ratio of the picture.
Width	Sets the width of a picture embedded in a header or footer

HeaderFooterPicture.Height

Description

Sets the height in pixels of a picture embedded within a header or footer

Signature: C#

```
public int Height
```

Signature: VB.NET

```
Public Function Height As Integer
```

Examples

C#

```
HeaderFooterPicture hfp = header.GetPicture();

//--- Set height of picture to 20 pixels
hfp.Height = 20;
```

vb.net

```
Dim hfp As HeaderFooterPicture = header.GetPicture()

'--- Set height of picture to 20 pixels
hfp.Height = 20
```

HeaderFooterPicture.LockAspectRatio

Description

Locks the aspect ratio of the picture.

Signature: C#

```
public bool LockAspectRatio
```

Signature: VB.NET

```
Public Function LockAspectRatio As Boolean
```

Remarks

Use this property to maintain or break the same aspect ratio for a picture in a header or footer while resizing. The default value is true.

Examples

C#

```
HeaderFooterPicture hfp = header.GetPicture();

//--- Lock the aspect ratio of the picture so that it is not
//--- stretched as it is resized
hfp.LockAspectRatio = true;
hfp.Height = 20;
hfp.Width = 30;
```

vb.net

```
Dim hfp As HeaderFooterPicture = header.GetPicture()

'--- Lock the aspect ratio of the picture so that it is not
'--- stretched as it is resized
hfp.LockAspectRatio = True
hfp.Height = 20
hfp.Width = 30
```

HeaderFooterPicture.Width

Description

Sets the width of a picture embedded in a header or footer

Signature: C#

```
public int Width
```

Signature: VB.NET

```
Public Function Width As Integer
```

Examples

C#

```
HeaderFooterPicture hfp = header.GetPicture();

//--- Set width of picture to 20 pixels
hfp.Width = 20;
```

vb.net

```
Dim hfp As HeaderFooterPicture = header.GetPicture()

'--- Set width of picture to 20 pixels
hfp.Width = 20
```

HeaderFooterSection

Introduced in build 4.5.1

Description

The HeaderFooterSection class represents an area of the header or footer. It allows you to set and retrieve the content in that area.

Signature: C#

```
public class HeaderFooterSection
```

Signature: VB.NET

```
Public Class HeaderFooterSection
```

Remarks

To retrieve a `HeaderFooterSection` object, call one of the following methods:

PageSetup.GetHeader(section)
PageSetup.GetFooter(section)

Methods

Name	Description
GetPicture()	Returns a Picture that has been embedded in the Header or Footer of the spreadsheet
GetText()	Returns the text that is contained in this section of the header or footer, including formatting codes
SetContent(String)	Sets the text that will appear in this section of the header or footer
SetContent(String, String)	Sets the text that will appear in this section of the header or footer and adds a picture
SetContent(String, System.IO.Stream)	Sets the text that will appear in this section of the header or footer and adds a picture

Nested Classes

Name	Description
Section	A Section value specifies a section of the header or footer.

HeaderFooterSection.GetPicture()

Description

Returns a Picture that has been embedded in the Header or Footer of the spreadsheet

Signature: C#

```
public HeaderFooterPicture GetPicture()
```

Signature: VB.NET

```
Public Function GetPicture() As HeaderFooterPicture
```

Returns

The Picture that has been embedded in the footer or header as a HeaderFooterPicture object

Examples

C#

```
HeaderFooterPicture pic = header.GetPicture();
```

vb.net

```
Dim pic As HeaderFooterPicture = header.GetPicture()
```

HeaderFooterSection.GetText()

Introduced in build 4.5.1

Description

Returns the text that is contained in this section of the header or footer, including formatting codes

Remarks

For information on the formatting codes used in headers and footers, see Formatting Headers and Footers

Signature: C#

```
public string GetText()
```

Signature: VB.NET

```
Public Function GetText() As String
```

HeaderFooterSection.Section

Introduced in build 4.5.1

Description

A Section value specifies a section of the header or footer.

Signature: C#

```
public sealed class Section : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Section
Inherits TypesafeEnum
```

Remarks

Use these values when calling PageSetup.GetHeader(Section) or PageSetup.GetFooter(Section) to specify which area of the header or footer you want to retrieve.

Values

Name	Description
Left	The left portion of the header or footer
Center	The center portion of the header or footer
Right	The right portion of the header or footer

HeaderFooterSection.SetContent

Overload	Description
SetContent(String, System.IO.Stream)	Sets the text that will appear in this section of the header or footer and adds a picture
SetContent(String, String)	Sets the text that will appear in this section of the header or footer and adds a picture
SetContent(String)	Sets the text that will appear in this section of the header or footer

HeaderFooterSection.SetContent(String, String)

Introduced in build 4.5.1

Description

Sets the text that will appear in this section of the header or footer and adds a picture

Signature: C#

```
public void SetContent(string text, string picture)
```

Signature: VB.NET

```
Public Sub SetContent(ByVal text As String, ByVal picture As String)
```

Remarks

For instructions on using the formatting codes for headers and footers, see Formatting Headers and Footers.

> ⚠ Total text, including formatting codes, for all sections of the header or footer cannot exceed 255 characters.

Parameters

text

The text to show in this section of the header or footer, including formatting codes.

> To insert the image defined by the **picture** parameter, you must include the **&G** formatting code in the text where you wish the picture to appear.

picture

The name of the picture to insert into the header, including relative path.

HeaderFooterSection.SetContent(String, System.IO.Stream)

Introduced in build 4.5.1

Description

Sets the text that will appear in this section of the header or footer and adds a picture

Signature: C#

```
public void SetContent(string text, System.IO.Stream picture)
```

Signature: VB.NET

```
Public Sub SetContent(ByVal text As String, ByVal picture As System.IO.Stream)
```

Remarks

For instructions on using the formatting codes for headers and footers, see Formatting Headers and Footers.

> Total text, including formatting codes, for all sections of the header or footer cannot exceed 255 characters.

Parameters

text

The text to show in this section of the header or footer, including formatting codes.

> To insert the image defined by the **picture** parameter, you must include the **&G** formatting code in the text where you wish the picture to appear.

picture

The picture to insert into the header as a System.IO.Stream object.

HeaderFooterSection.SetContent(String)

Introduced in build 4.5.1

Description

Sets the text that will appear in this section of the header or footer

Signature: C#

```
public void SetContent(System.String text)
```

Signature: VB.NET

```
Public Function SetContent(ByVal text As String)
```

Remarks

For instructions on using the formatting codes for headers and footers, see Formatting Headers and Footers.

Total text, including formatting codes, for all sections of the header or footer cannot exceed 255 characters.

Parameters

text

The text to show in this section of the header or footer, including formatting codes

Hyperlink

Description

Represents a hyperlink in an Area (which may contain only one cell). To create a Hyperlink object, call Cell.CreateHyperlink or Area.CreateHyperlink. To get an existing `Hyperlink`, use

Signature: C#

```
public sealed class Hyperlink
```

Signature: VB.NET

```
Public NotInheritable Class Hyperlink
```

Examples

C#

```
//--- Create a hyperlink
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          Cell cellA1 = ws.Cells["A1"];
          cellA1.Value = "SoftArtisans";
          Hyperlink link = cellA1.CreateHyperlink("http://www.softartisans.com");

          //--- Get an existing hyperlink
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Open("@C:\MySpreadsheet.xls");
          Worksheet ws = wb.Worksheets[0];
          Hyperlink firstLink = ws.Hyperlinks[0];
```

vb.net

```
'--- Create a hyperlink
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim cellA1 As Cell = ws.Cells("A1")
          cellA1.Value = "SoftArtisans"
          Dim link As Hyperlink = cellA1.CreateHyperlink("http://www.softartisans.com")

          '--- Get an existing hyperlink
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Hyperlink firstLink = ws.Hyperlinks(0)
```

Properties

Name	Description
Area	Returns the Area that the Hyperlink object is associated with. The set of cells that the `Area` object represents functions as a single hyperlink to the same href.
Href	Sets or returns the hyperlink's href value.
Text	Sets or returns the hyperlink's displayed text.

Hyperlink.Area

Description

Returns the Area that the Hyperlink object is associated with. The set of cells that the `Area` object represents functions as a single hyperlink to the same href.

Signature: C#

```
public Area Area{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Area() As Area
```

Examples

C#

```
Area linkArea = link.Area;
```

vb.net

```
Dim linkArea As Area = link.Area
```

Hyperlink.Href

Description

Sets or returns the hyperlink's href value.

Signature: C#

```
public System.String Href{ get; set; }
```

Signature: VB.NET

```
Public Property Href() As String
```

Examples

C#

```
//--- Get Href
          string linkHref = link.Href;

          //--- Set Href
          link.Href = "http://www.softartisans.com";
```

vb.net

```
'--- Get Href
          Dim linkHref As String = link.Href

          link.Href = "http://www.softartisans.com"
```

Hyperlink.Text

Description

Sets or returns the hyperlink's displayed text.

Signature: C#

```
public System.String Text{ get; set; }
```

Signature: VB.NET

```
Public Property Text() As String
```

Examples

C#

```
//--- Get Text
          string linkText = link.Text;

          //--- Set Text
          link.Text = "SoftArtisans";
```

vb.net

```
'--- Get Text
          Dim linkText As String = link.Text

          '--- Set Text
          link.Text = "SoftArtisans"
```

Hyperlinks

Description

The Hyperlinks collection contains all hyperlinks in a worksheet. The Worksheet.Hyperlinks property returns a `Hyperlinks` collection.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Hyperlinks : System.Collections.Generic.IEnumerable<Hyperlink>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Hyperlinks
  Implements System.Collections.Generic.IEnumerable(Of Hyperlink)
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Open(@"C:\MySpreadsheet.xls");
          Worksheet ws = wb.Worksheets[0];
          Hyperlinks links = ws.Hyperlinks;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Open("C:\MySpreadsheet.xls")
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim links As Hyperlinks = ws.Hyperlinks
```

Properties

Name	Description
Count	Returns the number of hyperlinks in the worksheet.

Indexers

Name	Description
Item(Int32)	Returns the hyperlink at the specified 0-based index in the collection of hyperlinks in the worksheet.

Methods

Name	Description
CreateHyperlink(Area, String)	Creates a new hyperlink in the specified area of cells. All cell values in the area will be converted to hyperlinks.
CreateHyperlink(Area, String, String)	Creates a new hyperlink in the specified area of cells. All cell values in the area will be converted to hyperlinks.
GetEnumerator()	Returns an IEnumerator for the Hyperlinks in a Worksheet

Hyperlinks.Count

Description

Returns the number of hyperlinks in the worksheet.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Examples

C#

```
int numLinks = links.Count;
```

vb.net

```
Dim numLinks As Integer = links.Count
```

Hyperlinks.CreateHyperlink

Overload	Description
CreateHyperlink(Area, String)	Creates a new hyperlink in the specified area of cells. All cell values in the area will be converted to hyperlinks.
CreateHyperlink(Area, String, String)	Creates a new hyperlink in the specified area of cells. All cell values in the area will be converted to hyperlinks.

Hyperlinks.CreateHyperlink(Area, String, String)

Description

Creates a new hyperlink in the specified area of cells. All cell values in the area will be converted to hyperlinks.

Signature: C#

```
public Hyperlink CreateHyperlink(Area area, System.String href, System.String text)
```

Signature: VB.NET

```
Public Function CreateHyperlink(ByVal area As Area, ByVal href As String, ByVal text As String) As
Hyperlink
```

Parameters

area

An Area object representing the cells that will be converted to hyperlinks.

href

The hyperlink's destination.

text

The text to display in the link.

Returns

A new Hyperlink object with the specified destination URL and display text.

Examples

C#

```
links.CreateHyperlink(a, "http://www.softartisans.com", "SoftArtisans");
```

vb.net

```
links.CreateHyperlink(a, "http://www.softartisans.com", "SoftArtisans")
```

Hyperlinks.CreateHyperlink(Area, String)

Description

Creates a new hyperlink in the specified area of cells. All cell values in the area will be converted to hyperlinks.

Signature: C#

```
public Hyperlink CreateHyperlink(Area area, System.String href)
```

Signature: VB.NET

```
Public Function CreateHyperlink(ByVal area As Area, ByVal href As String) As Hyperlink
```

Parameters

area

An Area object representing the cells that will be converted to hyperlinks.

href

The hyperlink's destination.

Returns

A new Hyperlink object with the specified destination URL.

Examples

C#

```
links.CreateHyperlink(a, "http://www.softartisans.com");
```

vb.net

```
links.CreateHyperlink(a, "http://www.softartisans.com")
```

Hyperlinks.GetEnumerator()

Description

Returns an IEnumerator for the Hyperlinks in a Worksheet

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the Hyperlinks in a Worksheet as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = ws.Hyperlinks.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = ws.Hyperlinks.GetEnumerator()
```

Hyperlinks.Item(Int32)

Description

Returns the hyperlink at the specified 0-based index in the collection of hyperlinks in the worksheet.

Signature: C#

```
public Hyperlink this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As Hyperlink
```

Parameters

index

The 0-based index of the hyperlink in the collection.

Returns

A Hyperlink object representing the hyperlink at the specified index in the collection of hyperlinks in the worksheet.

Remarks

Hyperlinks are not stored in the order in which they were created.

Examples

C#

```
Hyperlink link = links.Hyperlink[0];
```

vb.net

```
Dim link As Hyperlink = links.Hyperlink(0)
```

Legend

Description

A Legend object represents a chart's legend. To return `Legend` object, use Chart.Legend.

Signature: C#

```
public sealed class Legend : ChartRegion
```

Signature: VB.NET

```
Public NotInheritable Class Legend
  Inherits ChartRegion
```

Properties

Name	Description
Location	Sets or returns how the legend will be positioned in relation to the chart's plot area.

Vertical	Sets or returns whether the legend will be displayed vertically.
Visible	Sets or returns whether the chart legend will be displayed.

Nested Classes

Name	Description
LegendLocation	The LegendLocation class contains all available legend positions relative to a chart's plot area.

Legend.LegendLocation

Description

The LegendLocation class contains all available legend positions relative to a chart's plot area.

Signature: C#

```
public sealed class LegendLocation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class LegendLocation
  Inherits TypesafeEnum
```

Values

Name	Description
Bottom	The legend will be at the bottom of the plot area and horizontally centered.
Corner	The legend will be in the top-right corner of the plot area.
Left	The legend will be to the left of the chart and vertically centered.
NotDocked	Not docked
Right	The legend will be to the right of the chart and vertically centered.
Top	The legend will be at the top of the plot area and horizontally centered.

Legend.Location

Description

Sets or returns how the legend will be positioned in relation to the chart's plot area.

Signature: C#

```
public LegendLocation Location{ get; set; }
```

Signature: VB.NET

```
Public Property Location() As LegendLocation
```

Remarks

If you set Location to any position except NotDocked, the ChartRegion properties Height, Width, X, and Y will be automatically updated to reflect the new position.

If you manually set any of the ChartRegion properties Height, Width, X, or Y, the Location will be set to NotDocked.

Legend.Vertical

Description

Sets or returns whether the legend will be displayed vertically.

Signature: C#

```
public boolean Vertical{ get; set; }
```

Signature: VB.NET

```
Public Property Vertical() As Boolean
```

Remarks

If you change the value of Vertical, the ChartRegion properties Height, Width, X, and Y will be automatically updated to reflect the new position. Additionally, the Legend.Location property will be changed to NotDocked.

Legend.Visible

Description

Sets or returns whether the chart legend will be displayed.

Signature: C#

```
public boolean Visible{ get; set; }
```

Signature: VB.NET

```
Public Property Visible() As Boolean
```

NamedObject

Description

A `NamedObject` object may refer to an array, number, picture, or range. To create a `NamedObject` in an Excel file, use the corresponding method for the specific object you wish to create (Pictures.CreatePicture(), Workbook.CreateRange(), etc). To get a `NamedObject` from an existing file, use Workbook.GetNamedObject(), passing in the name of the object you wish to get.

Signature: C#

```
public sealed class NamedObject
```

Signature: VB.NET

```
Public NotInheritable Class NamedObject
```

Examples

The following sample gets a `NamedObject` from a Workbook.

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Open(@"C:\MySpreadsheet.xls");
          NamedObject namedObj = wb.GetNamedObject("MyNamedObject");
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Open("C:\MySpreadsheet.xls")
          Dim namedObj As NamedObject = wb.GetNamedObject("MyNamedObject")
```

Properties

Name	Description
Formula	If the NamedObject refers to a named range, Formula returns the formula associated with the range, e.g. "=Sheet1!A1:G10".
Name	Returns the name string that the specified NamedObject object represents.
NamedObjectType	Returns the type of object that the NamedObject refers to.
NameScope	Returns the scope of the name, relative to the Excel workbook.
Value	Returns the object referenced by this NamedObject. The object may be of any of the following types: Range, Picture, Array, Number. To determine the object type, use NamedObject.Type.

Nested Classes

Name	Description
BuiltInName	The BuiltInName class contains all built-in object names.
ReturnType	A ReturnType value specifies the type of object a NamedObject refers to.
Scope	A Scope value specifies the scope of the object that a NamedObject refers to.

NamedObject.BuiltInName

Description

The BuiltInName class contains all built-in object names.

Signature: C#

```
public sealed class BuiltInName : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BuiltInName
  Inherits TypesafeEnum
```

Values

Name	Description
Auto_Activate	Auto_Activate
Auto_Close	Auto_Close
Auto_Deactivate	Auto_Deactivate
Auto_Open	Auto_Open
Consolidate_Area	Consolidate_Area
Criteria	Criteria
Data_Form	Data_Form
Database	Database

Extract	Extract
FilterDatabase	FilterDatabase
Print_Area	Print_Area
Print_Titles	Print_Titles
Recorder	Recorder
Sheet_Title	Sheet_Title

NamedObject.Formula

Description

If the NamedObject refers to a named range, Formula returns the formula associated with the range, e.g. "=Sheet1!A1:G10".

Signature: C#

```
public System.String Formula{ get; }
```

Signature: VB.NET

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

Examples

C#

```
String rangeFormula = namedObj.Formula;
```

vb.net

```
Dim rangeFormula As String = namedObj.Formula
```

NamedObject.Name

Description

Returns the name string that the specified NamedObject object represents.

Signature: C#

```
public System.String Name{ get; }
```

Signature: VB.NET

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

Examples

C#

```
String objectName = namedObj.Name;
```

vb.net

```
Dim objectName As String = namedObj.Name
```

NamedObject.NamedObjectType

Description

Returns the type of object that the NamedObject refers to.

Signature: C#

```
public ReturnType NamedObjectType{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NamedObjectType() As ReturnType
```

Examples

C#

```
NamedObject.ReturnType type = namedObj.NameType;
```

vb.net

```
Dim type As NamedObject.ReturnType = namedObj.NameType
```

NamedObject.NameScope

Description

Returns the scope of the name, relative to the Excel workbook.

Signature: C#

```
public Scope NameScope{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NameScope() As Scope
```

Examples

C#

```
Object nameValue = namedObj.Value;
```

vb.net

```
Dim nameValue As Object = namedObj.Value
```

NamedObject.ReturnType

Description

A ReturnType value specifies the type of object a NamedObject refers to.

Signature: C#

```
public sealed class ReturnType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ReturnType
  Inherits TypesafeEnum
```

Values

Name	Description
Array	Array
ExternalReference	ExternalReference
Formula	Formula
Number	Number
Picture	Picture
Range	Range

NamedObject.Scope

Description

A Scope value specifies the scope of the object that a NamedObject refers to.

Signature: C#

```
public sealed class Scope : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Scope
  Inherits TypesafeEnum
```

Values

Name	Description
External	External
Workbook	Workbook
Worksheet	Worksheet

NamedObject.Value

Description

Returns the object referenced by this NamedObject. The object may be of any of the following types: Range, Picture, Array, Number. To determine the object type, use NamedObject.Type.

Signature: C#

```
public System.Object Value{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Value() As Object
```

Examples

C#

```
Object nameValue = namedObj.Value;
```

vb.net

```
Dim nameValue As Object = namedObj.Value
```

NamedStyle

Description

ExcelWriter creates three style types: GlobalStyle, NamedStyle, and CellStyle. All three style types derive from the Style class, and `NamedStyle` derives from `GlobalStyle`.

Signature: C#

```
public sealed class NamedStyle : GlobalStyle
```

Signature: VB.NET

```
Public NotInheritable Class NamedStyle
  Inherits GlobalStyle
```

Remarks

A style can be set or applied to cells, rows, columns, ranges, and areas. When a style is set , it is cloned and the object to which the style is assigned acquires all of that style's properties, including font proprties and number formatting. When a style is applied , only the differences between the new style and existing style properties (assigned through the ExcelWriter API or in Microsoft Excel) will take effect. For example, if the cell has a background color and the new style applied does not contain a background color, the cell's color will not be affected. However, if the new style includes a background color, it will replace the existing background color of the cell.

The ExcelApplication object contains a collection of `NamedStyle` s, which - unlike `GlobalStyle` s are accessible after the workbook is saved. This allows you to dynamically create a template, save it to disk, re-open it to populate it with data, and then apply your newly-created style. This collection of `NamedStyle` s includes Excel's built-in styles (such as the NORMAL style) and any user-defined styles in a workbook opened with the ExcelApplication object.

To create a `NamedStyle`, call Workbook.CreateNamedStyle.

To return a `NamedStyle`, call Workbook.GetNamedStyle.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();

          //--- Create a named style.
          Style myItalicStyle = wb.CreateNamedStyle("ItalicStyle");

          //--- Turn on italics for the named style.
          myGlobalStyle.Font.Italics = true;

          //--- Get a "total" cell
          Cell totalCell = wb.Worksheets[0].Cells[4, 4];

          //--- Check its value and apply style if necessary
          if(totalCell.Value > TARGET_TOTAL_VALUE)
               totalCell.ApplyStyle(myItalicStyle);
```

vb.net

```
Dim xla As New ExcelApplication()
          Workbook wb = xla.Create()

          '--- Create a named style.
          Dim myItalicStyle As Style = wb.CreateNamedStyle("ItalicStyle")

          '--- Turn on italics for the named style
          myGlobalStyle.Font.Italics = True

          '--- Get a "total" cell
          Dim totalCall As Cell = wb.Worksheets(0).Cells(4, 4)

          '--- Check its value and apply style if necessary
          If totalCell.Value > TARGET_TOTAL_VALUE Then
               totalCell.ApplyStyle(myItalicStyle)
          End If
```

Properties

Name	Description
Name	Returns the style's name. This can be used to access the style later, and will appear in the Styles dialog in Excel.

NamedStyle.Name

Description

Returns the style's name. This can be used to access the style later, and will appear in the Styles dialog in Excel.

Signature: C#

```
public System.String Name{ get; }
```

Signature: VB.NET

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

Examples

C#

```
String styleName = namedStyl.Name;
```

vb.net

```
Dim styleName As String = namedStyl.Name
```

NumberFormat

Description

NumberFormat is a helper class that creates custom number format strings, which are used for formatting numbers and dates. Use Style.NumberFormat to set a number format for a specific style.

Signature: C#

```
public sealed class NumberFormat
```

Signature: VB.NET

```
Public NotInheritable Class NumberFormat
```

Examples

C#

```
//--- Open a workbook.
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Open(@"C:\report.xls");

          //--- Create a style.
          Style percentStyle = wb.CreateStyle();

          //--- Specify that the number format for the style is a percentage with three
          //--- decimal places (0.000%).
          //--- The following line is equivalent to percentStyle.NumberFormat = "0.000%"
          percentStyle.NumberFormat = wb.NumberFormat.CreatePercentage(3);
```

vb.net

```
'--- Open a workbook.
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Open("C:\report.xls")

          '--- Create a style.
          Dim percentStyle As Style = wb.CreateStyle()

          '--- Specify that the number format for the style is a percentage with three
          '--- decimal places (0.000%).
          '--- The following line is equivalent to percentStyle.NumberFormat = "0.000%"
          percentStyle.NumberFormat = wb.NumberFormat.CreatePercentage(3)
```

Methods

Name	Description
CreateAccounting(Int32, Boolean, NumberFormat.Color)	Creates an accounting format string.

CreateCurrency(Int32, Boolean, Boolean, Boolean, NumberFormat.Color)	Creates a format string for displaying currency values.
CreateCurrency(Int32, Boolean, Boolean, NumberFormat.Color, String, String, Boolean, Boolean)	Creates a format string for displaying currency values, and allows locale-specific currency symbols.
CreateFraction(Int32)	Creates a string for displaying fractions.
CreateFraction(Int32, Int32)	Creates a string for displaying fractions.
CreateNumber(Int32, Boolean, Boolean, Boolean, Boolean, NumberFormat.Color)	Creates a format string for displaying numbers.
CreatePercentage(Int32)	Creates a string for displaying percentages.
CreateScientific(Int32)	Creates a scientific number format string.

Nested Classes

Name	Description
Color	The `Color` enumeration contains the colors that can be used to display negative numbers in a NumberFormat.
DateFormat	NumberFormat values for Date data types
Special_EN_US	NumberFormat values for specialized American numbers
TimeFormat	NumberFormat values for Time data types

NumberFormat.Color

Description

The `Color` enumeration contains the colors that can be used to display negative numbers in a NumberFormat.

Signature: C#

```
public sealed class Color : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Color
  Inherits TypesafeEnum
```

Values

Name	Description
Black	Negative values will be black.
Blue	Negative values will be blue.
Cyan	Negative values will be cyan.
Green	Negative values will be green.
Magenta	Negative values will be magenta.
None	Negative values will be the same color as positive values.
Red	Negative values will be red.
White	Negative values will be white.
Yellow	Negative values will be yellow.

NumberFormat.CreateAccounting(Int32, Boolean, NumberFormat.Color)

Description

Creates an accounting format string.

Signature: C#

```
public System.String CreateAccounting(int numDecimalPlaces, boolean useDollarSign, Color
negativeColor)
```

Signature: VB.NET

```
Public Function CreateAccounting(ByVal numDecimalPlaces As Integer, ByVal useDollarSign As Boolean,
ByVal negativeColor As Color) As String
```

Parameters

numDecimalPlaces

The number of decimal places to display.

useDollarSign

If `true`, the dollar sign will be displayed.

negativeColor

A color to use when displaying negative numbers. `negativeColor` may be null.

Returns

An accounting format string.

Examples

C#

```
//--- Create a style.
          Style styl = wb.CreateStyle();

          //--- The following creates the format
          //--- "_($* #,##0.00_);[Green]_($* #,##0.00_);_(* \"-\"_);_(@_)"
          styl.NumberFormat = wb.NumberFormat.CreateAccounting(2,
               true,
               NumberFormat.Color.Green);
```

vb.net

```
'--- Create a style.
          Dim styl As Style = wb.CreateStyle()

          '--- The following creates the format
          '--- "_($* #,##0.00_);[Green]_($* #,##0.00_);_(* \"-\"_);_(@_)"
          styl.NumberFormat = wb.NumberFormat.CreateAccounting(2, _
               True, _
               NumberFormat.Color.Green)
```

NumberFormat.CreateCurrency

Overload	Description
CreateCurrency(Int32, Boolean, Boolean, Boolean, NumberFormat.Color)	Creates a format string for displaying currency values.
CreateCurrency(Int32, Boolean, Boolean, NumberFormat.Color, String, String, Boolean, Boolean)	Creates a format string for displaying currency values, and allows locale-specific currency symbols.

NumberFormat.CreateCurrency(Int32, Boolean, Boolean, Boolean, NumberFormat.Color)

Description

Creates a format string for displaying currency values.

Signature: C#

```
public System.String CreateCurrency(int numDecimalPlaces, boolean useDollarSign, boolean
useNegativeSign, boolean useParensForNegatives, Color negativeColor)
```

Signature: VB.NET

```
Public Function CreateCurrency(ByVal numDecimalPlaces As Integer, ByVal useDollarSign As Boolean,
ByVal useNegativeSign As Boolean, ByVal useParensForNegatives As Boolean, ByVal negativeColor As
Color) As String
```

Parameters

numDecimalPlaces

The number of decimal places to display.

useDollarSign

If `true`, the dollar sign will be displayed.

useNegativeSign

If `true`, the negative sign will be used for negative values.

useParensForNegatives

If `true`, negative numbers will be shown in parentheses.

negativeColor

A color to use when displaying negative numbers. `negativeColor` may be null.

Returns

A currency format string.

Examples

C#

```
//--- Create a style.
          Style styl = wb.CreateStyle();

          //--- The following assigns the currency string "$#,##0.00;[Red]-$#,##0.00"
          //--- to the style.
          styl.NumberFormat =
               wb.NumberFormat.CreateCurrency(2,
               true,
               true,
               false,
               NumberFormat.Color.Red);
```

vb.net

```
'--- Create a style.
          Dim styl As Style = wb.CreateStyle()

          '--- The following assigns the currency string "$#,##0.00;[Red]-$#,##0.00"
          '--- to the style.
          styl.NumberFormat = wb.NumberFormat.CreateCurrency(2, _
               True, _
               True, _
               False, _
               NumberFormat.Color.Red)
```

NumberFormat.CreateCurrency(Int32, Boolean, Boolean, NumberFormat.Color, String, String, Boolean, Boolean)

Description

Creates a format string for displaying currency values, and allows locale-specific currency symbols.

Signature: C#

```
public System.String CreateCurrency(int numDecimalPlaces, boolean useNegativeSign, boolean
useParensForNegatives, Color negativeColor, System.String currencySymbol, System.String localeCode,
boolean currencyAtFront, boolean spaceBetween)
```

Signature: VB.NET

```
Public Function CreateCurrency(ByVal numDecimalPlaces As Integer, ByVal useNegativeSign As Boolean,
ByVal useParensForNegatives As Boolean, ByVal negativeColor As Color, ByVal currencySymbol As String,
ByVal localeCode As String, _
  ByVal currencyAtFront As Boolean, ByVal spaceBetween As Boolean) As String
```

Parameters

numDecimalPlaces

The number of decimal places to display.

useNegativeSign

If `true`, the negative sign will be used for negative values.

useParensForNegatives

If `true`, negative numbers will be shown in parentheses.

negativeColor

A color to use when displaying negative numbers. `negativeColor` may be null.

currencySymbol

A string representing the currency symbol. This could be the individual Unicode character for the currency symbol (such as the dollar, pound, euro, etc) or the alphabetic representation of the currency symbol (such as USD, GBP, EUR, etc). If this value is null, no currency string or formatting will be inserted.

localeCode

A string representing the Windows locale code, used for determining display on localized versions of Excel. This is optional, and can be null.

currencyAtFront

If the currencySymbol is non-null, this param determines if the currency string will appear before the number or after. If true, the symbol will appear before the value. If currencySymbol is null, this value is ignored.

spaceBetween

If the currencySymbol is non-null, and this param is true, a space will appear between the currency symbol and the value. If currencySymbol is null, this value is ignored.

Returns

A currency format string.

Examples

C#

```
//--- Create a style.
          Style styl = wb.CreateStyle();

          //--- The following assigns the currency string "[$€-2] #,##0.00;[Red][$€-2] #,##0.00"
          //--- to the style, which will use the Euro symbol prior to the values, with the
          //--- color red for negative values.
          styl.NumberFormat = wb.NumberFormat.CreateCurrency(2,
               false,
               false,
               NumberFormat.Color.Red,
               "€",
               "2",
               true,
               true);
```

vb.net

```
'--- Create a style.
          Dim styl As Style = wb.CreateStyle()

          '--- The following assigns the currency string "[$€-2] #,##0.00;[Red][$€-2] #,##0.00"
          '--- to the style, which will use the Euro symbol prior to the values, with the
          '--- color red for negative values.
          styl.NumberFormat = wb.NumberFormat.CreateCurrency(2, _
               False, _
               False, _
               NumberFormat.Color.Red, _
               "€", _
               "2", _
               True, _
               True)
```

NumberFormat.CreateFraction

Overload	Description
CreateFraction(Int32)	Creates a string for displaying fractions.
CreateFraction(Int32, Int32)	Creates a string for displaying fractions.

NumberFormat.CreateFraction(Int32, Int32)

Description

Creates a string for displaying fractions.

Signature: C#

```
public System.String CreateFraction(int numDigits, int denominator)
```

Signature: VB.NET

```
Public Function CreateFraction(ByVal numDigits As Integer, ByVal denominator As Integer) As String
```

Parameters

numDigits

The number of digits to display.

denominator

The value to use as denominator.

Returns

A fraction format string.

Examples

C#

```
//--- Create a style.
          Style styl = wb.CreateStyle();

          //--- Create the fraction format "# ??/10".
          styl.NumberFormat = wb.NumberFormat.CreateFraction(2, 10);
```

vb.net

```
'--- Create a style.
          Dim styl As Style = wb.CreateStyle()

          '--- Create the fraction format "# ??/10".
          styl.NumberFormat = wb.NumberFormat.CreateFraction(2, 10)
```

NumberFormat.CreateFraction(Int32)

Description

Creates a string for displaying fractions.

Signature: C#

```
public System.String CreateFraction(int numDigits)
```

Signature: VB.NET

```
Public Function CreateFraction(ByVal numDigits As Integer) As String
```

Parameters

numDigits

The number of digits to display.

Returns

A fraction format string.

Examples

C#

```
//--- Create a style.
          Style styl = wb.CreateStyle();

          //--- Create the fraction format "# ???/???".
          styl.NumberFormat = wb.NumberFormat.CreateFraction(3);
```

vb.net

```
'--- Create a style.
          Dim styl As Style = wb.CreateStyle()

          '--- Create the fraction format "# ???/???"
          styl.NumberFormat = wb.NumberFormat.CreateFraction(3)
```

NumberFormat.CreateNumber(Int32, Boolean, Boolean, Boolean, Boolean, NumberFormat.Color)

Description

Creates a format string for displaying numbers.

Signature: C#

```
public System.String CreateNumber(int numDecimalPlaces, boolean dropTrailingZeros, boolean
use1000Seperator, boolean useNegativeSign, boolean useParensForNegatives, Color negativeColor)
```

Signature: VB.NET

```
Public Function CreateNumber(ByVal numDecimalPlaces As Integer, ByVal dropTrailingZeros As Boolean,
ByVal use1000Seperator As Boolean, ByVal useNegativeSign As Boolean, ByVal useParensForNegatives As
Boolean, ByVal negativeColor As Color) As String
```

Parameters

numDecimalPlaces

The number of decimal places to display.

dropTrailingZeros

If `true`, trailing zeros will be dropped. If `false`, trailing zeros will be displayed.

use1000Seperator

If `true`, the 1000 separator (,) will be displayed.

useNegativeSign

If `true`, the negative sign will be used.

useParensForNegatives

If `true`, negative numbers will be shown in parentheses.

negativeColor

A color to use when displaying negative numbers. `negativeColor` may be null.

Returns

A number format string.

Examples

C#

```
//--- Create a style.
          Style styl = wb.CreateStyle();

          //--- The following assigns the number string "#,##0.##;-#,##0.##"
          //--- to the style.
          styl.NumberFormat =
               wb.NumberFormat.CreateNumber(2,
               true,
               true,
               true,
               false,
               NumberFormat.Color.Red);
```

vb.net

```
'--- Create a style.
          Dim styl As Style = wb.CreateStyle()

          '--- The following assigns the number string "#,##0.##;-#,##0.##"
          '--- to the style.
          styl.NumberFormat = _
               wb.NumberFormat.CreateNumber(2, _
               True, _
               True, _
               True, _
               False, _
               NumberFormat.Color.Redl)
```

NumberFormat.CreatePercentage(Int32)

Description

Creates a string for displaying percentages.

Signature: C#

```
public System.String CreatePercentage(int numDecimalPlaces)
```

Signature: VB.NET

```
Public Function CreatePercentage(ByVal numDecimalPlaces As Integer) As String
```

Parameters

numDecimalPlaces

The number of decimal places to display.

Returns

A format string for percentages.

Examples

C#

```
//--- Create a style.
          Style styl = wb.CreateStyle();

          //--- Create the percentage format "0.000%".
          styl.NumberFormat = wb.NumberFormat.CreatePercentage(3);
```

vb.net

```
'--- Create a style.
          Dim styl As Style = wb.CreateStyle()

          '--- Create the percentage format "0.000%".
          styl.NumberFormat = wb.NumberFormat.CreatePercentage(3)
```

NumberFormat.CreateScientific(Int32)

Description

Creates a scientific number format string.

Signature: C#

```
public System.String CreateScientific(int numDecimalPlaces)
```

Signature: VB.NET

```
Public Function CreateScientific(ByVal numDecimalPlaces As Integer) As String
```

Parameters

numDecimalPlaces

The number of decimal places to display.

Returns

A scientific number format string.

Examples

C#

```
//--- Create a style.
          Style styl = wb.CreateStyle();

          //--- Create the scientific format "0.000E+00".
          styl.NumberFormat = wb.NumberFormat.CreateScientific(3);
```

vb.net

```
'--- Create a style.
          Dim styl As Style = wb.CreateStyle()

          '--- Create the scientific format "0.000E+00".
          styl.NumberFormat = wb.NumberFormat.CreateScientific(3)
```

NumberFormat.DateFormat

Description

NumberFormat values for Date data types

Signature: C#

```
public sealed class DateFormat
```

Signature: VB.NET

```
Public NotInheritable Class DateFormat
```

Values

Name	Description
DayMonthYear	d-mmm-yyyy;@ For example, 2-Feb-08.)
DayOfWeekMonthDayYear	dddd, mmmm dd, yyyy (For example, Saturday, February 2, 2008.)
FullTextMonthDayYear	mmmm d, yyyy;@ (For example, February 2, 2008.)
FullTextMonthYear	mmmm-yy;@ (For example, February-08.)
MonthDay	m/d;@ (For example 2/2 for February 2nd.)
MonthDayYear	m/d/yy;@ (For example, 2/2/08.)
MonthDayYearHourMinute12	m/d/yy h:mm AM/PM;@ (For example, 2/2/08 12:00 AM)
MonthDayYearHourMinute24	m/d/yy h:mm;@ (For example, 2/2/08 0:00)
MonthDayYearPreserveLeadingZero	mm/dd/yy;@ (For example, 02/02/08.)
MonthSingleLetter	mmmmm;@ (For example, D for December or S for Septmeber.)
MonthSingleLetterYear	mmmmm-yy;@ (For example, F-08 for February 2008.)
TextDayMonth	d-mmm;@ (For example, 2-Feb.)
TextDayMonthYear	d-mmm-yy;@ (For example, 2-Feb-08.)
TextDayMonthYearPreserveLeadingZero	dd-mmm-yy;@ (For example, 02-Feb-08.)
TextMonthYear	mmm-yy;@ (For example, Feb-08.)

NumberFormat.Special_EN_US

Description

NumberFormat values for specialized American numbers

Signature: C#

```
public sealed class Special_EN_US
```

Signature: VB.NET

```
Public NotInheritable Class Special_EN_US
```

Values

Name	Description
PhoneNumber	[<=9999999]###-####;(###) ###-####
SocialSecurityNumber	000-00-0000
ZipCode	00000
ZipCodePlus4	00000-0000

NumberFormat.TimeFormat

Description

NumberFormat values for Time data types

Signature: C#

```
public sealed class TimeFormat
```

Signature: VB.NET

```
Public NotInheritable Class TimeFormat
```

Values

Name	Description
ElapsedTimeInHours	[h]:mm:ss;@ (For example, 846312:00:00. For elapsed time Microsoft uses the start date of January 1, 1900.)
ElapsedTimeInMinutes	[mm]:ss;@ (For example, 50778720:00. For elapsed time Microsoft uses the start date of January 1, 1900.)
ElapsedTimeInSeconds	[ss];@ (For example, 3046723200. For elapsed time Microsoft uses the start date of January 1, 1900.)
HourMinute12	h:mm AM/PM;@ (For example, 12:00 AM.)
HourMinute24	h:mm;@ (For example, 0:00.)
HourMinuteSecond12	h:mm:ss AM/PM;@ (For example, 12:00:00 AM.)
HourMinuteSecond24	h:mm:ss;@ (For example, 0:00:00.)
MinuteSecondTenths	mm:ss.0;@ (For example, 00:00.0.)

ObjectPositioning

Description

Enumeration of the available resize and reposition behaviors of an object when the cells in the spreadsheet it spans are changed.

Signature: C#

```
public sealed class ObjectPositioning : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ObjectPositioning
  Inherits TypesafeEnum
```

Values

Name	Description
MoveAndNoResize	Move the object and do not resizt it according to cell size
MoveAndResize	Move the object and resize it according to cell size
NoMoveAndNoResize	Do not move the object and do not change its size

Othersheet

Description

A Othersheet object represents a sheet that is not a worksheet or a chartsheet. This sheet represents VBA Module and dialog sheets. These are not currently implemented by ExcelWriter

Signature: C#

```
public sealed class Othersheet : Worksheet
```

Signature: VB.NET

```
Public NotInheritable Class Othersheet
  Inherits Worksheet
```

PageSetup

Description

The PageSetup class represents the page layout properties of a printed worksheet.

Signature: C#

```
public class PageSetup
```

Signature: VB.NET

```
Public Class PageSetup
```

Remarks

To return a `PageSetup` object, use Worksheet.PageSetup.

> ⚠ Since this object is tightly related to the client's printer setup, for those computers without a printer setup, changes to this object have no effect.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          PageSetup ps = wb.Worksheets[0].PageSetup;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ps As PageSetup = wb.Worksheets(0).PageSetup
```

Properties

Name	Description
BottomMargin	Sets or returns the value of the worksheet's bottom margin, in inches. This is the space between the worksheet data and the bottom edge of the printed page. This is set to 1 by default. The valid range of values is 0 through 100, and is also limited by the paper size.
CenterFooter	Sets or returns the text and formatting codes for the center footer that will appear on the printed worksheet. For instructions on using the formatting codes for headers and footers, see Formatting Headers and Footers.
CenterHeader	Sets or returns the text and formatting codes for the center header that will appear on the printed worksheet. For instructions on using the formatting codes for headers and footers, see Formatting Headers and Footers.
CenterHorizontally	Sets or returns whether or not the data of the worksheet is centered horizontally within the margins. This is is set to false by default.
CenterVertically	Sets or returns whether or not the data of the worksheet is centered vertically within the margins. This is is set to false by default.
FirstPageNumber	Sets or returns the first page number for the printed worksheet. By default, the first page number will be 1, or, if the print job does not start with page 1, this will be the number of the first page in the sequence of pages to print. Setting this property will set the FirstPageNumberAuto property to false.
FirstPageNumberAuto	Sets or returns whether Excel should determine the first page number for the printed worksheet. Setting the FirstPageNumber property will set this property to false.
FitToPagesTall	Sets or returns FitToPagesTall. This is set to "auto" by default. The valid range of values is 0 through 32767. A value of 0 means "auto".
FitToPagesWide	Sets or returns FitToPagesWide. This is set to 1 by default. The valid range of values is 0 through 32767. A value of 0 means "auto".
FooterMargin	Sets or returns the value of the footer margin in inches.
HeaderMargin	Sets or returns the value of the header margin in inches. This is the space between the header data and the top edge of the printed page. This is set to 0.5 by default. The valid range of values is 0 through 100, and is also limited by paper size.
LeftFooter	Sets or returns the text and formatting codes for the left footer that will appear on the printed worksheet.
LeftHeader	Sets or returns the text and formatting codes for the left header that will appear on the printed worksheet.
LeftMargin	Sets or returns the value of the left margin in inches. This is the space between the worksheet data and the left edge of the printed page. This is set to 0.75 by default. The valid range of values is 0 through 100, and is also limited by paper size.
Orientation	Sets or returns orientation of the printed worksheet.
PaperSize	Sets or returns the paper size that will be used when Excel renders and prints the worksheet.
PrintArea	Sets or returns the area of the worksheet that will be printed. Content outside of this area will not be printed.
PrintAsDraft	Sets or returns whether Excel will print the worksheet in draft quality. This can be used to reduce printing time. When this is set to `true`, Excel does not print gridlines and most graphics. This is set to false by default.
PrintCellErrorsAs	Sets or returns how cell errors will appear on the printed worksheet.
PrintComments	Sets or returns whether comments on the worksheet should be printed.
PrintCommentsAtEnd	Sets or returns whether comments will be printed at the end of the worksheet, or if they will be printed as they are displayed. If set to `true`, comments will be printed at the end.
PrintGridlines	Sets or returns whether the worksheet's horizontal and vertical cell gridlines will be printed.
PrintHeadings	Sets or returns whether the row and column headings will be printed for this worksheet.

PrintInBlackAndWhite	Sets or returns whether a worksheet formatted with colors will be printed in black and white. Set this to `true` if you are printing a colored worksheet on a black and white printer, or to reduce printing time on a color printer.
PrintOrder	Sets or returns the order in which data is numbered and printed when it does not fit on one page. The default order is down-then-over.
PrintTitles	Returns the Range of cells that contains the print titles. This range will contain up to two Area s, one for print title rows and another for the print title columns. These are the rows and columns that will be repeated on each printed page.
RightFooter	Sets or returns the text and formatting codes for the right footer that will appear on the printed worksheet.
RightHeader	Sets or returns the text and formatting codes for the right header that will appear on the printed worksheet.
RightMargin	Sets or returns the value of the right margin in inches. This is the space between the worksheet data and the right edge of the printed page. This is set to.75 by default. The valid range of values is 0 through 100, and is also limited by paper size.
TopMargin	Sets or returns the value for the top margin in inches. This is the space between the worksheet data and the top edge of the printed page. This is set to 1 by default. The valid range of values are 0 through 100, and is also limited by paper size.
UseZoom	Sets or returns whether the Zoom property or the FitToPagesTall and FitToPagesWide pair of properties will be used.
Zoom	Sets or returns the percentage by which the size of the printed worksheet is adjusted. The printed worksheet can be enlarged up to 400%, and reduced to 10%.

Methods

Name	Description
GetFooter(HeaderFooterSection.Section)	Returns a section of the footer.
GetHeader(HeaderFooterSection.Section)	Returns a section of the header.
SetPrintTitleColumns(Int32, Int32)	Sets the columns to use for the print titles. These are the columns that will be repeated on each page of a printed worksheet.
SetPrintTitleRows(Int32, Int32)	Sets the rows to use for the print titles. These are the rows that will be repeated on each page of a printed worksheet.

Nested Classes

Name	Description
CellErrorsAs	A CellErrorsAs value determines how cell errors will appear in the printed worksheet.
PageOrientation	PageOrientation specifies the page orientation of the printed worksheet.
PagePaperSize	PagePaperSize specifies the paper size for printing.
PagePrintOrder	`PagePrintOrder` specifies the order in which a worksheet should be printed.

PageSetup.BottomMargin

Description

Sets or returns the value of the worksheet's bottom margin, in inches. This is the space between the worksheet data and the bottom edge of the printed page. This is set to 1 by default. The valid range of values is 0 through 100, and is also limited by the paper size.

Signature: C#

```
public double BottomMargin{ get; set; }
```

Signature: VB.NET

```
Public Property BottomMargin() As Double
```

Examples

C#

```
//--- Get BottomMargin
          double btmMargin = ps.BottomMargin;

          //--- Set BottomMargin
          ps.BottomMargin = 2;
```

vb.net

```
'--- Get BottomMargin
          Dim btmMargin As Double = ps.BottomMargin

          '--- Set BottomMargin
          ps.BottomMargin = 2
```

PageSetup.CellErrorsAs

Description

A CellErrorsAs value determines how cell errors will appear in the printed worksheet.

Signature: C#

```
public sealed class CellErrorsAs : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class CellErrorsAs
  Inherits TypesafeEnum
```

Values

Name	Description
Blank	[blank]
Dash	--
Displayed	Displayed
NA	#NA

PageSetup.CenterFooter

Description

Sets or returns the text and formatting codes for the center footer that will appear on the printed worksheet. For instructions on using the formatting codes for headers and footers, see Formatting Headers and Footers.

Signature: C#

```
public System.String CenterFooter{ get; set; }
```

Signature: VB.NET

```
Public Property CenterFooter() As String
```

Remarks

Total text for all sections of the footer – including formatting codes – cannot exceed 255 characters.

Examples

C#

```
//--- Get CenterFooter
          string ctrFooter = ps.CenterFooter;

          //--- Set CenterFooter
          ps.CenterFooter = "&\"Tahoma\"&BCenter Footer&B";
```

vb.net

```
'--- Get CenterFooter
          Dim ctrFooter As String = ps.CenterFooter

          '--- Set CenterFooter
          ps.CenterFooter = "&""Tahoma""&BCenter Footer&B"
```

PageSetup.CenterHeader

Description

Sets or returns the text and formatting codes for the center header that will appear on the printed worksheet. For instructions on using the formatting codes for headers and footers, see Formatting Headers and Footers.

Signature: C#

```
public System.String CenterHeader{ get; set; }
```

Signature: VB.NET

```
Public Property CenterHeader() As String
```

Remarks

Note: Total text, including formatting codes, for all sections of the header cannot exceed 255 characters.

Examples

C#

```
//--- Get CenterHeader
          string ctrHeader = ps.CenterHeader

          //--- Set CenterHeader
          ps.CenterHeader = "&\"Tahoma\"&BCenter Header&B";
```

vb.net

```
'--- Get CenterHeader
          Dim ctrHeader As String = ps.CenterHeader

          '--- Set CenterHeader
          ps.CenterHeader = "&""Tahoma""&BCenter Header&B"
```

PageSetup.CenterHorizontally

Description

Sets or returns whether or not the data of the worksheet is centered horizontally within the margins. This is is set to false by default.

Signature: C#

```
public boolean CenterHorizontally{ get; set; }
```

Signature: VB.NET

```
Public Property CenterHorizontally() As Boolean
```

Examples

C#

```
//--- Get CenterHorizontally
          bool centerHoriz = ps.CenterHorizontally

          //--- Set CenterHorizontally
          ps.CenterHorizontally = true;
```

vb.net

```
'--- Get CenterHorizontally
          Dim centerHoriz As Boolean = ps.CenterHorizontally

          '--- Set CenterHorizontally
          ps.CenterHorizontally = True
```

PageSetup.CenterVertically

Description

Sets or returns whether or not the data of the worksheet is centered vertically within the margins. This is is set to false by default.

Signature: C#

```
public boolean CenterVertically{ get; set; }
```

Signature: VB.NET

```
Public Property CenterVertically() As Boolean
```

Examples

C#

```
//--- Get CenterVertically
          bool centerVert = ps.CenterVertically;

          //--- Set CenterVertically
          ps.CenterVertically = true;
```

vb.net

```
'--- Get CenterVertically
          Dim centerVert As Boolean = ps.CenterVertically

          '--- Set CenterVertically
          ps.CenterVertically = True
```

PageSetup.FirstPageNumber

Description

Sets or returns the first page number for the printed worksheet. By default, the first page number will be 1, or, if the print job does not start with page 1, this will be the number of the first page in the sequence of pages to print. Setting this property will set the FirstPageNumberAuto property to false.

Signature: C#

```
public int FirstPageNumber{ get; set; }
```

Signature: VB.NET

```
Public Property FirstPageNumber() As Integer
```

Examples

C#

```
//--- Get FirstPageNumber
          int firstPage = ps.FirstPageNumber;

          //--- Set FirstPageNumber
          ps.FirstPageNumber = 3;
```

vb.net

```
'--- Get FirstPageNumber
          Dim firstPage As Integer = ps.FirstPageNumber

          '--- Set FirstPageNumber
          ps.FirstPageNumber = 3
```

PageSetup.FirstPageNumberAuto

Description

Sets or returns whether Excel should determine the first page number for the printed worksheet. Setting the FirstPageNumber property will set this property to false.

Signature: C#

```
public boolean FirstPageNumberAuto{ get; set; }
```

Signature: VB.NET

```
Public Property FirstPageNumberAuto() As Boolean
```

Examples

C#

```
//--- Get FirstPageNumberAuto
          bool firstPageAuto = ps.FirstPageNumberAuto;

          //--- Set FirstPageNumberAuto
          ps.FirstPageNumberAuto = true;
```

vb.net

```
'--- Get FirstPageNumberAuto
          Dim firstPageAuto As Boolean = ps.FirstPageNumberAuto

          '--- Set FirstPageNumberAuto
          ps.FirstPageNumberAuto = False
```

PageSetup.FitToPagesTall

Description

Sets or returns FitToPagesTall. This is set to "auto" by default. The valid range of values is 0 through 32767. A value of 0 means "auto".

Signature: C#

```
public int FitToPagesTall{ get; set; }
```

Signature: VB.NET

```
Public Property FitToPagesTall() As Integer
```

Remarks

Note: The pair of properties FitToPagesWide and FitToPagesTall, and Zoom are mutually exclusive.

Examples

C#

```
//--- Get FitToPagesTall
          int pagesTall = ps.FitToPagesTall;

          //--- Set FitToPagesTell
          ps.FitToPagesTall = 2;
```

vb.net

```
'--- Get FitToPagesTall
          Dim pagesTall As Integer = ps.FitToPagesTall

          '--- Set FitPagesToTall
          ps.FitToPagesTall = 2
```

PageSetup.FitToPagesWide

Description

Sets or returns FitToPagesWide. This is set to 1 by default. The valid range of values is 0 through 32767. A value of 0 means "auto".

Signature: C#

```
public int FitToPagesWide{ get; set; }
```

Signature: VB.NET

```
Public Property FitToPagesWide() As Integer
```

Remarks

Note: The pair of properties FitToPagesWide and FitToPagesTall, and Zoom are mutually exclusive.

Examples

C#

```
//--- Get FitToPagesWide
          bool pagesWide = ps.FitToPagesWide;

          //--- Set FitToPagesWide
          ps.FitToPagesWide = 2;
```

vb.net

```
'--- Get FitToPagesWide
          Dim pagesWide As Boolean = ps.FitToPagesWide

          '--- Set FitToPagesWide
          ps.FitToPagesWide = 2
```

PageSetup.FooterMargin

Description

Sets or returns the value of the footer margin in inches.

Signature: C#

```
public double FooterMargin{ get; set; }
```

Signature: VB.NET

```
Public Property FooterMargin() As Double
```

Remarks

This is the space between the footer data and the bottom edge of the printed page. This is set to.5 by default. The valid range of values is 0 through 100, and is also limited by the paper size.

Examples

C#

```
//--- Get FooterMargin
          double ftrMargin = ps.FooterMargin;

          //--- Set FooterMargin
          ps.FooterMargin = 1;
```

vb.net

```
'--- Get FooterMargin
          Dim ftrMargin As Double = ps.FooterMargin

          '--- Set FooterMargin
          ps.FooterMargin = 1
```

PageSetup.GetFooter(HeaderFooterSection.Section)

Introduced in build 4.5.1

Description

Returns a section of the footer.

Signature: C#

```
public HeaderFooterSection GetFooter(HeaderFooterSection.Section section)
```

Signature: VB.NET

```
Public Function GetFooter(ByVal section As HeaderFooterSection.Section) As HeaderFooterSection
```

Parameters

section

The section of the footer to return

Returns

A HeaderFooterSection object representing the specified section of the footer

PageSetup.GetHeader(HeaderFooterSection.Section)

Introduced in build 4.5.1

Description

Returns a section of the header.

Signature: C#

```
public HeaderFooterSection GetHeader(HeaderFooterSection.Section section)
```

Signature: VB.NET

```
Public Function GetHeader(ByVal section As HeaderFooterSection.Section) As HeaderFooterSection
```

Parameters

section

The section of the header to return

Returns

A HeaderFooterSection object representing the specified section of the header

PageSetup.HeaderMargin

Description

Sets or returns the value of the header margin in inches. This is the space between the header data and the top edge of the printed page. This is set to 0.5 by default. The valid range of values is 0 through 100, and is also limited by paper size.

Signature: C#

```
public double HeaderMargin{ get; set; }
```

Signature: VB.NET

```
Public Property HeaderMargin() As Double
```

Examples

C#

```
//--- Get HeaderMargin
          double hdrMargin = ps.HeaderMargin;

          //--- Set HeaderMargin
          ps.HeaderMargin = 1;
```

vb.net

```
'--- Get HeaderMargin
          Dim hdrMargin As Double = ps.HeaderMargin

          '--- Set HeaderMargin
          ps.HeaderMargin = 1
```

PageSetup.LeftFooter

Description

Sets or returns the text and formatting codes for the left footer that will appear on the printed worksheet.

Signature: C#

```
public System.String LeftFooter{ get; set; }
```

Signature: VB.NET

```
Public Property LeftFooter() As String
```

Remarks

For instructions on using the formatting codes for headers and footers, see Headers and Footers.

> ⚠ Total text, including formatting codes, for all sections of the footer cannot exceed 255 characters.

Examples

C#

```
//--- Get LeftFooter
          string lftFooter = ps.LeftFooter;

          //--- Set LeftFooter
          ps.LeftFooter = "&\"Tahoma\"&BLeft Footer&B";
```

vb.net

```
'--- Get LeftFooter
          Dim lftFooter As String = ps.LeftFooter

          '--- Set LeftFooter
          ps.LeftFooter = "&""Tahoma""&BLeft Footer&B"
```

PageSetup.LeftHeader

Description

Sets or returns the text and formatting codes for the left header that will appear on the printed worksheet.

Signature: C#

```
public System.String LeftHeader{ get; set; }
```

Signature: VB.NET

```
Public Property LeftHeader() As String
```

Remarks

For instructions on using the formatting codes for headers and footers, see Headers and Footers.

> ⚠ Total text, including formatting codes, for all sections of the header cannot exceed 255 characters.

Examples

C#

```
//--- Get LeftHeader
          string lftHeader = ps.LeftHeader;

          //--- Set LeftHeader
          ps.LeftHeader = "&\"Tahoma\"&BLeft Header&B";
```

vb.net

```
'--- Get LeftHeader
          Dim lftHeader As String = ps.LeftHeader

          '--- Set LeftHeader
          ps.Leftheader = "&""Tahoma""&BLeft Header&B"
```

PageSetup.LeftMargin

Description

Sets or returns the value of the left margin in inches. This is the space between the worksheet data and the left edge of the printed page. This is set to 0.75 by default. The valid range of values is 0 through 100, and is also limited by paper size.

Signature: C#

```
public double LeftMargin{ get; set; }
```

Signature: VB.NET

```
Public Property LeftMargin() As Double
```

Examples

C#

```
//--- Get LeftMargin
          double lftMargin = ps.LeftMargin;

          //--- Set LeftMargin
          ps.LeftMargin = 1;
```

vb.net

```
'--- Get LeftMargin
          Dim lftMargin As Double = ps.LeftMargin

          '--- Set LeftMargin
          ps.LeftMargin = 1
```

PageSetup.Orientation

Description

Sets or returns orientation of the printed worksheet.

Signature: C#

```
public PageOrientation Orientation{ get; set; }
```

Signature: VB.NET

```
Public Property Orientation() As PageOrientation
```

Examples

C#

```
//--- Get Orientation
          PageSetup.PageOrientation orientation = ps.Orientation

          //--- Set Orientation
          ps.Orientation = PageSetup.PageOrientation.Landscape;
```

vb.net

```
'--- Get Orientation
          Dim orientation As PageSetup.PageOrientation = ps.Orientation

          '--- Set Orientation
          ps.Orientation = PageSetup.PageOrientation.Landscape
```

PageSetup.PageOrientation

Description

PageOrientation specifies the page orientation of the printed worksheet.

Signature: C#

```
public sealed class PageOrientation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class PageOrientation
  Inherits TypesafeEnum
```

Values

Name	Description
Landscape	Landscape
Portrait	Portrait

PageSetup.PagePaperSize

Description

PagePaperSize specifies the paper size for printing.

Signature: C#

```
public sealed class PagePaperSize : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class PagePaperSize
  Inherits TypesafeEnum
```

Values

Name	Description
A3	A3
A4	A4
A5	A5
A4Small	A4 Small
B4	B4
B5	B5
CSheet	C Sheet
Default	Default
DSheet	D Sheet
Envelope9	Envelope 9
Envelope10	Envelope 10

Envelope11	Envelope 11
Envelope12	Envelope 12
Envelope14	Envelope 14
EnvelopeB4	Envelope B4
EnvelopeB5	Envelope B5
EnvelopeB6	Envelope B6
EnvelopeC3	Envelope C3
EnvelopeC4	Envelope C4
EnvelopeC5	Envelope C5
EnvelopeC6	Envelope C6
EnvelopeC65	Envelope C65
EnvelopeDL	Envelope DL
EnvelopeItaly	Envelope Italy
EnvelopeMonarch	Envelope Monarch
EnvelopePersonal	Envelope Personal
ESheet	E Sheet
Executive	Executive
FanfoldLegalGerman	Fanfold Legal German
FanfoldStdGerman	Fanfold Standard German
FanfoldUS	Fanfold US
Folio	Folio
Ledger	Ledger
Legal	Legal
Letter	Letter
LetterSmall	Letter Small
Note	Note
Paper10x14	10x14
Paper11x17	11x17
Quarto	Quarto
Statement	Statement
Tabloid	Tabloid
User	User

PageSetup.PagePrintOrder

Description

`PagePrintOrder` specifies the order in which a worksheet should be printed.

Signature: C#

```
public sealed class PagePrintOrder : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class PagePrintOrder
  Inherits TypesafeEnum
```

Values

Name	Description
DownThenOver	Down then over
OverThenDown	Over then down

PageSetup.PaperSize

Description

Sets or returns the paper size that will be used when Excel renders and prints the worksheet.

Signature: C#

```
public PagePaperSize PaperSize{ get; set; }
```

Signature: VB.NET

```
Public Property PaperSize() As PagePaperSize
```

Examples

C#

```
//--- Get PaperSize
          PageSetup.PagePaperSize size = ps.PaperSize;

          //--- Set PaperSize
          ps.PaperSize = PageSetup.PagePaperSize.A4;
```

vb.net

```
'--- Get PaperSize
          Dim size As PageSetup.PagePaperSize = ps.PaperSize

          '--- Set PaperSize
          ps.PaperSize = PageSetup.PagePaperSize.A4
```

PageSetup.PrintArea

Description

Sets or returns the area of the worksheet that will be printed. Content outside of this area will not be printed.

Signature: C#

```
public Area PrintArea{ get; set; }
```

Signature: VB.NET

```
Public Property PrintArea() As Area
```

Examples

C#

```
//--- Get PrintArea
          Area prntArea = ps.PrintArea;

          //--- Set PrintArea
          ps.PrintArea = prntArea;
```

vb.net

```
'--- Get PrintArea
          Dim prntArea As Area = ps.PrintArea

          '--- Set PrintArea
          ps.PrintArea = prntArea
```

PageSetup.PrintAsDraft

Description

Sets or returns whether Excel will print the worksheet in draft quality. This can be used to reduce printing time. When this is set to `true`, Excel does not print gridlines and most graphics. This is set to false by default.

Signature: C#

```
public boolean PrintAsDraft{ get; set; }
```

Signature: VB.NET

```
Public Property PrintAsDraft() As Boolean
```

Examples

C#

```
//--- Get PrintAsDraft
          bool draft = ps.PrintAsDraft;

          //--- Set PrintAsDraft
          ps.PrintAsDraft = true;
```

vb.net

```
'--- Get PrintAsDraft
          Dim draft As Boolean = ps.PrintAsDraft

          '--- Set PrintAsDraft
          ps.PrintAsDraft = True
```

PageSetup.PrintCellErrorsAs

Description

Sets or returns how cell errors will appear on the printed worksheet.

Signature: C#

```
public CellErrorsAs PrintCellErrorsAs{ get; set; }
```

Signature: VB.NET

```
Public Property PrintCellErrorsAs() As CellErrorsAs
```

Examples

C#

```
//--- Get PrintCellErrorAs
          PageSetup.CellErrorsAs errorsAs = ps.PrintCellErrorsAs;

          //--- Set PrintCellErrorAs
          ps.PrintCellErrorsAs = PageSetup.CellErrorsAs.Dash;
```

vb.net

```
'--- Get PrintCellErrorAs
          Dim errorsAs As PageSetup.CellErrorsAs = ps.PrintCellErrorsAs

          '--- Set PrintCellErrorAs
          ps.PrintCellErrorsAs = PageSetup.CellErrorsAs.Dash
```

PageSetup.PrintComments

Description

Sets or returns whether comments on the worksheet should be printed.

Signature: C#

```
public boolean PrintComments{ get; set; }
```

Signature: VB.NET

```
Public Property PrintComments() As Boolean
```

Remarks

If the comments should be printed at the end of the worksheet, then set the PrintCommentsAtEnd property to true, otherwise the comments will be printed as they are displayed.

Examples

C#

```
//--- Get PrintComments
          bool prntComments = ps.PrintComments

          //--- Set PrintComments
          ps.PrintComments = true;
```

vb.net

```
'--- Get PrintComments
          Dim prntComments As Boolean = ps.PrintComments

          '--- Set PrintComments
          ps.PrintComments = True
```

PageSetup.PrintCommentsAtEnd

Description

Sets or returns whether comments will be printed at the end of the worksheet, or if they will be printed as they are displayed. If set to `true`, comments will be printed at the end.

Signature: C#

```
public boolean PrintCommentsAtEnd{ get; set; }
```

Signature: VB.NET

```
Public Property PrintCommentsAtEnd() As Boolean
```

Examples

C#

```
//--- Get PrintCommentsAtEnd
          bool commentsAtEnd = ps.PrintCommentsAtEnd;

          //--- Set PrintCommentsAtEnd
          ps.PrintCommentsAtEnd = true;
```

vb.net

```
'--- Get PrintCommentsAtEnd
          Dim commentsAtEnd As Boolean = ps.PrintCommentsAtEnd

          '--- Set PrintCommentsAtEnd
          ps.PrintCommentsAtEnd = True
```

PageSetup.PrintGridlines

Description

Sets or returns whether the worksheet's horizontal and vertical cell gridlines will be printed.

Signature: C#

```
public boolean PrintGridlines{ get; set; }
```

Signature: VB.NET

```
Public Property PrintGridlines() As Boolean
```

Examples

C#

```
//--- Get PrintGridlines
          bool prntGridlines = ps.PrintGridlines

          //--- Set PrintGridlines
          ps.PrintGridlines = true;
```

vb.net

```
'--- Get PrintGridlines
          Dim prntGridlines As Boolean = ps.PrintGridlines

          '--- Set PrintGridlines
          ps.PrintGridlines = True
```

PageSetup.PrintHeadings

Description

Sets or returns whether the row and column headings will be printed for this worksheet.

Signature: C#

```
public boolean PrintHeadings{ get; set; }
```

Signature: VB.NET

```
Public Property PrintHeadings() As Boolean
```

Examples

C#

```
//--- Get PrintHeadings
          bool prntHeadings = ps.PrintHeadings;

          //--- Set PrintHeadings
          ps.PrintHeadings = true;
```

vb.net

```
'--- Get PrintHeadings
          Dim prntHeadings As Boolean = ps.PrintHeadings

          '--- Set PrintHeadings
          ps.PrintHeadings = True
```

PageSetup.PrintInBlackAndWhite

Description

Sets or returns whether a worksheet formatted with colors will be printed in black and white. Set this to `true` if you are printing a colored worksheet on a black and white printer, or to reduce printing time on a color printer.

Signature: C#

```
public boolean PrintInBlackAndWhite{ get; set; }
```

Signature: VB.NET

```
Public Property PrintInBlackAndWhite() As Boolean
```

Examples

C#

```
//--- Get PrintBlackAndWhite
          bool bw = ps.PrintInBlackAndWhite;

          //--- Set PrintBlackAndWhite
          ps.PrintInBlackAndWhite = true;
```

vb.net

```
'--- Get PrintBlackAndWhite
          Dim bw As Boolean = ps.PrintInBlackAndWhite

          '--- Set PrintBlackAndWhite
          ps.PrintInBlackAndWhite = True
```

PageSetup.PrintOrder

Description

Sets or returns the order in which data is numbered and printed when it does not fit on one page. The default order is down-then-over.

Signature: C#

```
public PagePrintOrder PrintOrder{ get; set; }
```

Signature: VB.NET

```
Public Property PrintOrder() As PagePrintOrder
```

Examples

C#

```
//--- Get PrintOrder
          PageSetup.PagePrintOrder order = ps.PrintOrder;

          //--- Set PrintOrder
          ps.PrintOrder = PageSetup.PagePrintOrder.OverThenDown;
```

vb.net

```
'--- Get PrintOrder
          Dim order As PageSetup.PagePrintOrder = ps.PrintOrder

          '--- Set PrintOrder
          ps.PrintOrder = PageSetup.PagePrintOrder.OverThenDown
```

PageSetup.PrintTitles

Description

Returns the Range of cells that contains the print titles. This range will contain up to two Area s, one for print title rows and another for the print title columns. These are the rows and columns that will be repeated on each printed page.

Signature: C#

```
public Range PrintTitles{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PrintTitles() As Range
```

Examples

C#

```
Range prntTitles = ps.PrintTitles;
```

vb.net

```
Dim prntTitles As Range = ps.PrintTitles
```

PageSetup.RightFooter

Description

Sets or returns the text and formatting codes for the right footer that will appear on the printed worksheet.

Signature: C#

```
public System.String RightFooter{ get; set; }
```

Signature: VB.NET

```
Public Property RightFooter() As String
```

Remarks

For instructions on using the formatting codes for headers and footers, see Headers and Footers.

> ⚠ Total text, including formatting codes, for all sections of the footer cannot exceed 255 characters.

Examples

C#

```
//--- Get RightFooter
          string rghtFooter = ps.RightFooter;

          //--- Set RightFooter
          ps.RightFooter = "&\"Tahoma\"&BRight Footer&B";
```

vb.net

```
'--- Get RightFooter
          Dim rghtFooter As String = ps.RightFooter

          '--- Set RightFooter
          ps.RightFooter = "&""Tahoma""&BRight Footer&B"
```

PageSetup.RightHeader

Description

Sets or returns the text and formatting codes for the right header that will appear on the printed worksheet.

Signature: C#

```
public System.String RightHeader{ get; set; }
```

Signature: VB.NET

```
Public Property RightHeader() As String
```

Remarks

For instructions on using the formatting codes for headers and footers, see Headers and Footers.

> ⚠ Total text, including formatting codes, for all sections of the header cannot exceed 255 characters.

Examples

C#

```
//--- Get RightHeader
          string rghtHeader = ps.RightHeader;

          //--- Set RightHeader
          ps.RightHeader = "&\"Tahoma\"&BRight Header&B";
```

vb.net

```
'--- Get RightHeader
          Dim rghtHeader As String = ps.RightHeader

          '--- Set RightHeader
          ps.RightHeader = "&""Tahoma""&BRight Header&B"
```

PageSetup.RightMargin

Description

Sets or returns the value of the right margin in inches. This is the space between the worksheet data and the right edge of the printed page. This is set to.75 by default. The valid range of values is 0 through 100, and is also limited by paper size.

Signature: C#

```
public double RightMargin{ get; set; }
```

Signature: VB.NET

```
Public Property RightMargin() As Double
```

Examples

C#

```
//--- Get RightMargin
          double rghtMargin = ps.RightMargin;

          //--- Set RightMargin
          ps.RightMargin = 1;
```

vb.net

```
'--- Get RightMargin
          Dim rghtMargin As Double = ps.RightMargin

          '--- Set RightMargin
          ps.RightMargin = 1
```

PageSetup.SetPrintTitleColumns(Int32, Int32)

Description

Sets the columns to use for the print titles. These are the columns that will be repeated on each page of a printed worksheet.

Signature: C#

```
public void SetPrintTitleColumns(int firstColumn, int numColumns)
```

Signature: VB.NET

```
Public Sub SetPrintTitleColumns(ByVal firstColumn As Integer, ByVal numColumns As Integer)
```

Parameters

firstColumn

The 0-based number of the first column to in the print title.

numColumns

The number of columns to include in the print title.

Examples

C#

```
ps.SetPrintTitleColumns(0, 5);
```

vb.net

```
ps.SetPrintTitleColumns(0, 5)
```

PageSetup.SetPrintTitleRows(Int32, Int32)

Description

Sets the rows to use for the print titles. These are the rows that will be repeated on each page of a printed worksheet.

Signature: C#

```
public void SetPrintTitleRows(int firstRow, int numRows)
```

Signature: VB.NET

```
Public Sub SetPrintTitleRows(ByVal firstRow As Integer, ByVal numRows As Integer)
```

Parameters

firstRow

The 0-based number of the first row to in the print title.

numRows

The number of rows to include in the print title.

Examples

C#

```
ps.SetPrintTitleRows(0, 2);
```

vb.net

```
ps.SetPrintTitleRows(0, 2)
```

PageSetup.TopMargin

Description

Sets or returns the value for the top margin in inches. This is the space between the worksheet data and the top edge of the printed page. This is set to 1 by default. The valid range of values are 0 through 100, and is also limited by paper size.

Signature: C#

```
public double TopMargin{ get; set; }
```

Signature: VB.NET

```
Public Property TopMargin() As Double
```

Examples

C#

```
//--- Get TopMargin
          double tpMargin = ps.TopMargin;

          //--- Set TopMargin
          ps.TopMargin = 1.5;
```

vb.net

```
'--- Get TopMargin
          Dim tpMargin As Double = ps.TopMargin

          '--- Set TopMargin
          ps.TopMargin = 1.5
```

PageSetup.UseZoom

Description

Sets or returns whether the Zoom property or the FitToPagesTall and FitToPagesWide pair of properties will be used.

Signature: C#

```
public boolean UseZoom{ get; set; }
```

Signature: VB.NET

```
Public Property UseZoom() As Boolean
```

Examples

C#

```
//--- Get UseZoom
          bool useZm = ps.UseZoom;

          //--- Set Zoom
          ps.UseZoom = false;
```

vb.net

```
'--- Get UseZoom
          Dim useZm As Boolean = ps.UseZoom

          '--- Set UseZoom
          ps.UseZoom = False
```

PageSetup.Zoom

Description

Sets or returns the percentage by which the size of the printed worksheet is adjusted. The printed worksheet can be enlarged up to 400%, and reduced to 10%.

Signature: C#

```
public int Zoom{ get; set; }
```

Signature: VB.NET

```
Public Property Zoom() As Integer
```

Examples

C#

```
//--- Get Zoom
          int zm = ps.Zoom;

          //--- Set Zoom
          ps.Zoom = 75;
```

vb.net

```
'--- Get Zoom
          Dim zm As integer = ps.Zoom

          '--- Set Zoom
          ps.Zoom = 75
```

Palette

Description

The Palette class represents Excel's color palette.

Signature: C#

```
public sealed class Palette
```

Signature: VB.NET

```
Public NotInheritable Class Palette
```

Remarks

Each Excel document has a single color palette that manages the colors for formatting and some drawing layer colors. A color palette contains 56 colors that may be referenced. Each of these 56 colors may be customized, but changing a color in the palette will affect all formatting that uses the referenced color. A workbook's color palette is accessible through Workbook.Palette.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Palette pal = wb.Palette;
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim pal As Palette = wb.Palette
```

Fields

Name	Description
MinColorIndex	The minimum valid index of a color within the palette.
MinRGBValue	The minimum value that an RGB component value may have.
MaxColorIndex	The maximum valid index of a color within the palette.
MaxRGBValue	The maximum value that an RGB component value may have.

Methods

Name	Description
GetClosestColor(Int32, Int32, Int32)	Returns the color in the palette closest to the specified RGB value. The closest match is defined as the color in the palette that has the minimum difference between the specified RGB value and its RGB value.
GetClosestColor(System.Drawing.Color)	Returns the color in the palette closest to the specified color. The closest match is defined as the color in the palette that has the minimum difference between RGB value of the specified color and its RGB value.
GetColor(Int32, Int32, Int32)	Returns the color in the palette that has the specified RGB value. An exception will be thrown if no color in the palette has the specified RGB value.
GetColor(System.Drawing.Color)	Returns the color in the palette identical to the specified color. An exception will be thrown if the color is not in the palette.
GetColorAt(Int32)	Returns the color at the specified 0-based index within the Palette.
SetColor(Color, Int32, Int32, Int32)	Changes the RGB value of a specified `Color` object. `Color` must exist in the palette.
SetColorAt(Int32, Int32, Int32, Int32)	Replaces the color at the specified index with a new color.

Nested Classes

Name	Description
SystemColor	Primary System Colors

Palette.GetClosestColor

Overload	Description
GetClosestColor(Int32, Int32, Int32)	Returns the color in the palette closest to the specified RGB value. The closest match is defined as the color in the palette that has the minimum difference between the specified RGB value and its RGB value.
GetClosestColor(System.Drawing.Color)	Returns the color in the palette closest to the specified color. The closest match is defined as the color in the palette that has the minimum difference between RGB value of the specified color and its RGB value.

Palette.GetClosestColor(Int32, Int32, Int32)

Description

Returns the color in the palette closest to the specified RGB value. The closest match is defined as the color in the palette that has the minimum difference between the specified RGB value and its RGB value.

Signature: C#

```
public Color GetClosestColor(int red, int green, int blue)
```

Signature: VB.NET

```
Public Function GetClosestColor(ByVal red As Integer, ByVal green As Integer, ByVal blue As Integer)
As Color
```

Parameters

red

The red component of the desired color. Must be between 0 and 255.

green

The green component of the desired color. Must be between 0 and 255.

blue

The blue component of the desired color. Must be between 0 and 255.

Returns

The closest color in the palette to the specified RGB value

Examples

C#

```
Color clr = pal.GetClosestColor(162, 221, 139);
```

vb.net

```
Dim clr As Color = pal.GetCloestColor(162, 221, 139)
```

Palette.GetClosestColor(System.Drawing.Color)

Description

Returns the color in the palette closest to the specified color. The closest match is defined as the color in the palette that has the minimum difference between RGB value of the specified color and its RGB value.

Signature: C#

```
public Color GetClosestColor(System.Drawing.Color color)
```

Signature: VB.NET

```
Public Function GetClosestColor(ByVal color As System.Drawing.Color) As Color
```

Parameters

color

The desired color.

Returns

The closest color in the palette to the specified RGB value

Examples

C#

```
Color clr = pal.GetClosestColor(System.Drawing.Color.Red);
```

vb.net

```
Dim clr As Color = pal.GetClosestColor(System.Drawing.Color.Red)
```

Palette.GetColor

Overload	Description
GetColor(Int32, Int32, Int32)	Returns the color in the palette that has the specified RGB value. An exception will be thrown if no color in the palette has the specified RGB value.
GetColor(System.Drawing.Color)	Returns the color in the palette identical to the specified color. An exception will be thrown if the color is not in the palette.

Palette.GetColor(Int32, Int32, Int32)

Description

Returns the color in the palette that has the specified RGB value. An exception will be thrown if no color in the palette has the specified RGB value.

Signature: C#

```
public Color GetColor(int red, int green, int blue)
```

Signature: VB.NET

```
Public Function GetColor(ByVal red As Integer, ByVal green As Integer, ByVal blue As Integer) As Color
```

Parameters

red

The red component of the desired color. Must be between 0 and 255.

green

the green component of the desired color. Must be between 0 and 255.

blue

the blue component of the desired color. Must be between 0 and 255.

Returns

A `Color` object representing the color with the specified RGB value.

Examples

C#

```
Color blue = pal.GetColor(0, 0, 255);
```

vb.net

```
Dim blue As Color = pal.GetColor(0, 0, 255)
```

Palette.GetColor(System.Drawing.Color)

Description

Returns the color in the palette identical to the specified color. An exception will be thrown if the color is not in the palette.

Signature: C#

```
public Color GetColor(System.Drawing.Color color)
```

Signature: VB.NET

```
Public Function GetColor(ByVal color As System.Drawing.Color) As Color
```

Parameters

color

The desired color.

Returns

A `Color` object in the palette that matches the specified color.

Examples

C#

```
Color clr = pal.GetColor(System.Drawing.Color.Red);
```

vb.net

```
Dim clr As Color = pal.GetColor(System.Drawing.Color.Red)
```

Palette.GetColorAt(Int32)

Description

Returns the color at the specified 0-based index within the Palette.

Signature: C#

```
public Color GetColorAt(int index)
```

Signature: VB.NET

```
Public Function GetColorAt(ByVal index As Integer) As Color
```

Parameters

index

The 0-based index of the color to return. Must be between 0 and 55.

Returns

The color at the specified index.

Examples

C#

```
Color clr = pal.GetColorAt(4);
```

vb.net

```
Dim clr As Color = pal.GetColorAt(4)
```

Palette.MaxColorIndex

Description

The maximum valid index of a color within the palette.

Signature: C#

```
public static const int MaxColorIndex = 55;
```

Signature: VB.NET

```
Public Shared Const MaxColorIndex As Integer = 55
```

Palette.MaxRGBValue

Description

The maximum value that an RGB component value may have.

Signature: C#

```
public static const int MaxRGBValue = 255;
```

Signature: VB.NET

```
Public Shared Const MaxRGBValue As Integer = 255
```

Palette.MinColorIndex

Description

The minimum valid index of a color within the palette.

Signature: C#

```
public static const int MinColorIndex = 0;
```

Signature: VB.NET

```
Public Shared Const MinColorIndex As Integer = 0
```

Palette.MinRGBValue

Description

The minimum value that an RGB component value may have.

Signature: C#

```
public static const int MinRGBValue = 0;
```

Signature: VB.NET

```
Public Shared Const MinRGBValue As Integer = 0
```

Palette.SetColor(Color, Int32, Int32, Int32)

Description

Changes the RGB value of a specified `Color` object. `Color` must exist in the palette.

Signature: C#

```
public void SetColor(Color color, int red, int green, int blue)
```

Signature: VB.NET

```
Public Sub SetColor(ByVal color As Color, ByVal red As Integer, ByVal green As Integer, ByVal blue As
Integer)
```

Parameters

color

The `Color` object to modify. This must be a Palette color (i.e. Color.IsPaletteColor must return true).

red

The red component of the desired color. Must be between 0 and 255.

green

The green component of the desired color. Must be between 0 and 255.

blue

The blue component of the desired color. Must be between 0 and 255.

Examples

C#

```
palette.SetColor(clr, 162, 221, 138);
```

vb.net

```
palette.SetColor(clr, 162, 221, 138)
```

Palette.SetColorAt(Int32, Int32, Int32, Int32)

Description

Replaces the color at the specified index with a new color.

Signature: C#

```
public Color SetColorAt(int index, int red, int green, int blue)
```

Signature: VB.NET

```
Public Function SetColorAt(ByVal index As Integer, ByVal red As Integer, ByVal green As Integer, ByVal blue As Integer) As Color
```

Parameters

index

The 0-based index of the color to modify. Must be between 0 and 55.

red

The red component of the desired color. Must be between 0 and 255.

green

The green component of the desired color. Must be between 0 and 255.

blue

The blue component of the desired color. Must be between 0 and 255.

Returns

A `Color` object representing the new color at the specified index in the palette.

Examples

C#

```
pal.SetColorAt(4, 162, 221, 138);
```

vb.net

```
pal.SetColorAt(4, 162, 221, 138)
```

Palette.SystemColor

Description

Primary System Colors

Signature: C#

```
public sealed class SystemColor
```

Signature: VB.NET

```
Public NotInheritable Class SystemColor
```

Values

Name	Description
AutomaticFill	Default color for fills and backgrounds, depends on system color settings when opening the file, usually white
AutomaticLine	Default color for lines, depends on system color settings when opening the file, usually Black
Black	Black. RGB: 0,0,0; Hex: #000000
Blue	Blue. RGB: 0, 0, 255; Hex: #0000FF
Cyan	Cyan. RGB: 0, 255, 255; Hex: #00FFFF
Green	Green. RGB: 0, 255, 0; Hex: #00FF000
Magenta	Magenta. RGB: 255, 0, 255; Hex: #FF00FF
Red	Red. RGB: 255, 0, 0; Hex: #FF0000
White	White. RGB: 255, 255, 255; Hex: #FFFFFF
Yellow	Yellow. RGB: 255, 255, 0; Hex: #FFFF00

Pattern

Description

Pattern contains all available background patterns for charts and pictures.

Signature: C#

```
public sealed class Pattern : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Pattern
  Inherits TypesafeEnum
```

Remarks

Note that the pattern you select will appear in front of the object's background color. The examples in the following table assume a white background color.

Additionally, the pattern names do not match up cleanly with the names of the patterns available for ChartInterior objects. The corresponding names are listed below.

Values

Name	Description (Style)	Description (ChartInterior)
DiagonalCrosshatch	Diagonal crosshatch pattern	Small checker board
DiagonalStripe	Diagonal stripe pattern.	Dark upward diagonal

Gray6	6.25-percent grey pattern.	10%
Gray12	12.5-percent grey pattern.	20%
Gray25	25-percent gray pattern.	25%
Gray50	50-percent gray pattern.	50%
Gray75	75-percent gray pattern.	70%
HorizontalStripe	Horizontal stripe pattern.	Dark horizontal
None	No pattern type applied.	No pattern
ReverseDiagonalStripe	Reverse diagonal stripe patter	Dark downward diagonal
Solid	Solid pattern type.	No pattern (uses background color)
ThickDiagonalCrosshatch	Thick diagonal crosshatch pattern.	Trellis
ThinDiagonalCrosshatch	Thin diagonal crosshatch pattern.	30%
ThinDiagonalStripe	Thin diagonal stripe pattern.	Light upward diagonal
ThinHorizontalCrosshatch	Thin horizontal crosshatch pattern.	Small grid
ThinHorizontalStripe	Thin horizontal stripe pattern.	Light horizontal
ThinReverseDiagonalStripe	Thin reverse diagonal stripe pattern.	Light downward diagonal
ThinVerticalStripe	Thin vertical stripe pattern.	Light vertical
VerticalStripe	Vertical stripe pattern.	Dark vertical

Picture

Description

A Picture object represents a picture in a worksheet and allows the size, position, and content of the picture to be managed. Once created, the data of a picture may not be changed or replaced with the data of another picture. Instead, delete the picture and create a new one.

Signature: C#

```
public sealed class Picture : Shape
```

Signature: VB.NET

```
Public NotInheritable Class Picture
  Inherits Shape
```

Remarks

To create a `Picture` object, call Pictures.CreatePicture.

Properties

Name	Description

Height	Sets or returns the height of the Picture, in points (1/72 inch).
LockAspectRatio	Sets or returns whether the aspect ratio of the picture will be locked in Excel. This setting has no effect within ExcelWriter.
Name	Sets or returns the picture's name.
PictureData	Returns a new byte array containing the data of the picture as it would appear in an image file.
Position	Sets or returns the Picture 's position in the worksheet.
Shape	Returns the underlying Shape object for this Picture, allowing advanced formatting to be set.
Width	Sets or returns the width of the Picture, in points (1/72 inch).

Picture.Height

Description

Sets or returns the height of the Picture, in points (1/72 inch).

Signature: C#

```
public double Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Double
```

Picture.LockAspectRatio

Description

Sets or returns whether the aspect ratio of the picture will be locked in Excel. This setting has no effect within ExcelWriter.

Signature: C#

```
public boolean LockAspectRatio{ get; set; }
```

Signature: VB.NET

```
Public Property LockAspectRatio() As Boolean
```

Picture.Name

Description

Sets or returns the picture's name.

Signature: C#

```
public System.String Name{ get; set; }
```

Signature: VB.NET

```
Public Property Name() As String
```

Picture.PictureData

Description

Returns a new byte array containing the data of the picture as it would appear in an image file.

Signature: C#

```
public System.SByte[] PictureData{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PictureData() As SByte()
```

Picture.Position

Description

Sets or returns the Picture 's position in the worksheet.

Signature: C#

```
public Anchor Position{ get; set; }
```

Signature: VB.NET

```
Public Property Position() As Anchor
```

Picture.Shape

Description

Returns the underlying Shape object for this Picture, allowing advanced formatting to be set.

Signature: C#

```
public Shape Shape{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Shape() As Shape
```

Picture.Width

Description

Sets or returns the width of the Picture, in points (1/72 inch).

Signature: C#

```
public double Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Double
```

Pictures

Description

The Pictures collection represents the set of pictures in a worksheet. To return a `Pictures` collection, use Worksheet.Pictures.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Pictures : System.Collections.Generic.IEnumerable<Picture>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Pictures
  Implements System.Collections.Generic.IEnumerable(Of Picture)
```

Properties

Name	Description
Count	Returns the number of Picture objects in the collection.

Indexers

Name	Description
Item(Int32)	Returns the Picture object at the specified position in the Pictures collection. `Picture` is the indexer for the `Pictures` class.
Item(String)	Returns the Picture object with the specified name in the Pictures collection. `Picture` is the indexer for the `Pictures` class.

Methods

Name	Description
CreatePicture(String, Anchor)	Creates a new Picture object from an existing image.
CreatePicture(System.IO.Stream, Anchor)	Creates a new Picture object from an existing image.
GetEnumerator()	Returns an IEnumerator for the Pictures in a Worksheet
ListPictures()	Returns an array of all the Picture objects in the worksheet.
Remove(Int32)	Deletes the specified Picture object from the worksheet.
Remove(Picture)	Deletes the specified Picture object from the worksheet.

Pictures.Count

Description

Returns the number of Picture objects in the collection.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Pictures.CreatePicture

Overload	Description
CreatePicture(String, Anchor)	Creates a new Picture object from an existing image.

CreatePicture(System.IO.Stream, Anchor)	Creates a new Picture object from an existing image.

Pictures.CreatePicture(String, Anchor)

Description

Creates a new Picture object from an existing image.

Signature: C#

```
public Picture CreatePicture(System.String pictureFile, Anchor position)
```

Signature: VB.NET

```
Public Function CreatePicture(ByVal pictureFile As String, ByVal position As Anchor) As Picture
```

Parameters

pictureFile

The file to create the picture from. This file should be in JPEG, PNG, BMP, or GIF format and accessible in the context ExcelWriter is running in.

position

The position to place the upper-left corner of the new picture.

Returns

A `Picture` object.

Pictures.CreatePicture(System.IO.Stream, Anchor)

Description

Creates a new Picture object from an existing image.

Signature: C#

```
public Picture CreatePicture(System.IO.Stream pictureStream, Anchor position)
```

Signature: VB.NET

```
Public Function CreatePicture(ByVal pictureStream As System.IO.Stream, ByVal position As Anchor) As
Picture
```

Parameters

pictureStream

A stream containing data for a JPEG, PNG, BMP, or GIF formatted image. The calling application is responsible for closing this stream.

position

The position to place the upper-left corner of the new picture.

Returns

A `Picture` object.

Pictures.GetEnumerator()

Description

Returns an IEnumerator for the Pictures in a Worksheet

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the Pictures in a Worksheet as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = ws.Pictures.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = ws.Pictures.GetEnumerator()
```

Pictures.Item

Overload	Description
Item(Int32)	Returns the Picture object at the specified position in the Pictures collection. `Picture` is the indexer for the `Pictures` class.
Item(String)	Returns the Picture object with the specified name in the Pictures collection. `Picture` is the indexer for the `Pictures` class.

Pictures.Item(Int32)

Description

Returns the Picture object at the specified position in the Pictures collection. `Picture` is the indexer for the `Pictures` class.

Signature: C#

```
public Picture this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As Picture
```

Parameters

index

A valid index between 0 and Pictures.Count.

Returns

The `Picture` object at the specified position.

Pictures.Item(String)

Description

Returns the Picture object with the specified name in the Pictures collection. `Picture` is the indexer for the `Pictures` class.

Signature: C#

```
public Picture this[System.String name] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal name As String) As Picture
```

Parameters

name

A name assigned by Excel to a Picture.

Returns

The `Picture` object with the specified name.

Pictures.ListPictures()

Description

Returns an array of all the Picture objects in the worksheet.

Signature: C#

```
public Picture[] ListPictures()
```

Signature: VB.NET

```
Public Function ListPictures() As Picture()
```

Returns

An array of all the `Picture` objects in the workbook.

Pictures.Remove

Overload	Description
Remove(Int32)	Deletes the specified Picture object from the worksheet.
Remove(Picture)	Deletes the specified Picture object from the worksheet.

Pictures.Remove(Int32)

Description

Deletes the specified Picture object from the worksheet.

Signature: C#

```
public void Remove(int position)
```

Signature: VB.NET

```
Public Sub Remove(ByVal position As Integer)
```

Parameters

position

The 0-based position of the `Picture` object in the array returned by Pictures.ListPictures.

Pictures.Remove(Picture)

Description

Deletes the specified Picture object from the worksheet.

Signature: C#

```
public void Remove(Picture picture)
```

Signature: VB.NET

```
Public Sub Remove(ByVal picture As Picture)
```

Parameters

picture

The `Picture` object to remove.

Range

Description

A Range object represents a range in a workbook.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Range
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Range
```

Remarks

A range is a collection of areas. An area is a rectangular collection of cells. The areas in a range may be non-adjacent, and a range can include areas in different worksheets.

To create a `Range` (without a name), call one of the following methods:

- Workbook.CreateRange(rangeFormula)
- Worksheet.CreateRange(rangeFormula)

If a range is named, it will be accessible when the workbook is opened in Microsoft Excel. To create a named range, call one of the following methods:

- Workbook.CreateNamedRange(rangeFormula, rangeName)
- Worksheet.CreateNamedRange(firstRow, firstColumn, numRows, numColumns, rangeName)
- Worksheet.CreateNamedRange(rangeFormula, rangeName)

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          wb.Worksheets.CreateWorksheet("Sheet2");
          Range rng = wb.CreateRange("=Sheet1!A1:A3, Sheet2!A2:C5");
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          wb.Worksheets.CreateWorksheet("Sheet2")
          Dim rng As Range = wb.CreateRange("=Sheet1!A1:A3, Sheet2!A2:C5")
```

Properties

Name	Description
AreaCount	Returns the number of Area objects contained in this Range.
Areas	Returns an array of the rectangular areas contained in the Range.
BorderAround	Returns a Border object that represents a border around the range.
FirstCellStyle	Returns the style for the first cell in the range. Changes to this style will apply only to the first cell. To assign this style to the entire Range, set Range.SetStyle() to this Style reference.

Indexers

Name	Description
Item(Int32)	Returns the Area object at the specified 0-based index.

Methods

Name	Description
ApplyStyle(Style)	Applies a style to the area. When a style is applied - rather than set (see Range.SetStyle) - only the differences between the new style and style properties previously assigned to the range (through the ExcelWriter API) will take effect. For example, if the range has a background color and the new style applied does not contain a background color, the area's color will not be affected.
ClearContent()	Clears the content of all cells in the range. Calling this method is equivalent to setting an range's style to "Normal" and its cell values to null.
JoinRange(Range)	Adds another range to this range.
RemoveConditionalFormat()	This method removes any ConditionalFormat objects from the Range.
SetConditionalFormat(ConditionalFormat)	This method copies the specified ConditionaFormat object and associates it with this Range. If any ConditionalFormat objects already exist within the Range, they will be removed and replaced by the specifed one.
SetDataValidation(DataValidation)	Assigns a data validation rule to all cells in the Range. If the dataValidation object uses a local reference or area in a formula (i.e. "=A5:B7"), then the range will adjust those local references accordingly based on the range's location. The original dataValidation object will not be effected.
SetStyle(Style)	Sets the style for every cell in this range. When a style is set - rather than applied (see Range.ApplyStyle) - all previously assigned style properties, including font and number formatting, will be overwritten for all cells in the range.
ToString()	Returns a string representation of the Area object.

Range.ApplyStyle(Style)

Description

Applies a style to the area. When a style is applied - rather than set (see Range.SetStyle) - only the differences between the new style and style properties previously assigned to the range (through the ExcelWriter API) will take effect. For example, if the range has a background color and the new style applied does not contain a background color, the area's color will not be affected.

Signature: C#

```
public void ApplyStyle(Style style)
```

Signature: VB.NET

```
Public Sub ApplyStyle(ByVal style As Style)
```

Parameters

style

A Style object representing the style to apply to the cell.

Range.AreaCount

Description

Returns the number of Area objects contained in this Range.

Signature: C#

```
public int AreaCount{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property AreaCount() As Integer
```

Examples

C#

```
int numAreas = rng.AreaCount;
```

vb.net

```
Dim numAreas As Integer = rng.AreaCount
```

Range.Areas

Description

Returns an array of the rectangular areas contained in the Range.

Signature: C#

```
public Area[] Areas{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Areas() As Area()
```

Examples

C#

```
Area[] rectAreas = rng.Areas;
```

vb.net

```
Dim rectAreas() As Area = rng.Areas
```

Range.BorderAround

Description

Returns a Border object that represents a border around the range.

Signature: C#

```
public Border BorderAround{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property BorderAround() As Border
```

Examples

C#

```
//--- Get BorderAround
          Border b = rng.BorderAround;

          //--- Set BorderAround
          rng.BorderAround = b;
```

vb.net

```
'--- Get BorderAround
          Dim b As Border = rng.BorderAround

          '--- Set BorderAround
          rng.BorderAround = b
```

Range.ClearContent()

Description

Clears the content of all cells in the range. Calling this method is equivalent to setting an range's style to "Normal" and its cell values to null.

Signature: C#

```
public void ClearContent()
```

Signature: VB.NET

```
Public Sub ClearContent()
```

Examples

C#

```
rng.ClearContent();
```

vb.net

```
rng.ClearContent()
```

Range.FirstCellStyle

Description

Returns the style for the first cell in the range. Changes to this style will apply only to the first cell. To assign this style to the entire Range, set Range.SetStyle() to this Style reference.

Signature: C#

```
public Style FirstCellStyle{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property FirstCellStyle() As Style
```

Examples

C#

```
Style firstStyle = rng.FirstCellStyle;
```

vb.net

```
Dim firstStyle As Style = rng.FirstCellStyle
```

Range.Item(Int32)

Description

Returns the Area object at the specified 0-based index.

Signature: C#

```
public Area this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As Area
```

Parameters

index

The index of the `Area` to get.

Returns

The `Area` at the specified 0-based index.

Examples

C#

```
Area a = rng[0];
```

vb.net

```
Dim a As Area = rng(0)
```

Range.JoinRange(Range)

Description

Adds another range to this range.

Signature: C#

```
public void JoinRange(Range range)
```

Signature: VB.NET

```
Public Sub JoinRange(ByVal range As Range)
```

Parameters

range

The range to be added to the current range.

Examples

C#

```
rng.JoinRange(rng2);
```

vb.net

```
rng.JoinRange(rng2)
```

Range.RemoveConditionalFormat()

Description

This method removes any ConditionalFormat objects from the Range.

Signature: C#

```
public void RemoveConditionalFormat()
```

Signature: VB.NET

```
Public Sub RemoveConditionalFormat()
```

Examples

C#

```
range.RemoveConditionalFormat();
```

vb.net

```
range.RemoveConditionalFormat()
```

Range.SetConditionalFormat(ConditionalFormat)

Description

This method copies the specified ConditionaFormat object and associates it with this Range. If any ConditionalFormat objects already exist within the Range, they will be removed and replaced by the specifed one.

Signature: C#

```
public void SetConditionalFormat(ConditionalFormat conditionalFormat)
```

Signature: VB.NET

```
Public Sub SetConditionalFormat(ByVal conditionalFormat As ConditionalFormat)
```

Parameters

conditionalFormat

Examples

C#

```
range.SetConditionalFormat(condFmt);
```

vb.net

```
range.SetConditionalFormat(condFmt)
```

Range.SetDataValidation(DataValidation)

Description

Assigns a data validation rule to all cells in the Range. If the dataValidation object uses a local reference or area in a formula (i.e. "=A5:B7"), then the range will adjust those local references accordingly based on the range's location. The original dataValidation object will not be effected.

Signature: C#

```
public void SetDataValidation(DataValidation dataValidation)
```

Signature: VB.NET

```
Public Sub SetDataValidation(ByVal dataValidation As DataValidation)
```

Parameters

dataValidation

A DataValidation object representing the data validation rule to assign to the Range.

Examples

C#

```
rng.SetDataValidation(myDataValidation);
```

vb.net

```
rng.SetDataValidation(myDataValidation)
```

Range.SetStyle(Style)

Description

Sets the style for every cell in this range. When a style is set - rather than applied (see Range.ApplyStyle) - all previously assigned style properties, including font and number formatting, will be overwritten for all cells in the range.

Signature: C#

```
public void SetStyle(Style style)
```

Signature: VB.NET

```
Public Sub SetStyle(ByVal style As Style)
```

Parameters

style

The Style object to assign to the range.

Examples

C#

```
rng.SetStyle(styl);
```

vb.net

```
rng.SetStyle(styl)
```

Range.ToString()

Description

Returns a string representation of the Area object.

Signature: C#

```
public virtual System.String ToString()
```

Signature: VB.NET

```
Public Overridable Function ToString() As String
```

Returns

A string representation of the `Area` object.

Examples

C#

```
string color = area.ToString();
```

vb.net

```
Dim color As String = area.ToString()
```

RowProperties

Description

The RowProperties class represents a single row in a worksheet. To return a `RowProperties` object, use Worksheet.GetRowProperties.

Signature: C#

```
public sealed class RowProperties
```

Signature: VB.NET

```
Public NotInheritable Class RowProperties
```

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets[0];
          RowProperties rowProps = ws.GetRowProperties(0);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)
          Dim rowProps As RowProperties = ws.GetRowProperties(0)
```

Properties

Name	Description
Height	Sets or returns the height of the row in points (1/72 of an inch).
Hidden	Sets or returns whether or not the row will be hidden when the workbook is opened in Excel.
OutlineCollapsed	Sets or returns whether the row's outlined row group is collapsed. If the outline is collapsed, the row is hidden.
OutlineLevel	Sets or returns the outline level of the row, as a value between 0 and 7. The outline level is used to control grouping in Excel. For example, to group several rows, set their outline levels to 1; to group a subset of rows within the group, set their outline levels to 2.
RowNumber	Returns the row's 0-based index.
Style	Sets or returns a Style object that determines how cells in the row look, including number format, font, orientation, etc. Accessing this style will not affect the formatting of other cells with the same global style.

Methods

Name	Description
ApplyStyle(Style)	Applies a style to the row.
AutoFitHeight()	Sets the width of the row to the height of the tallest cell value in the row.

RowProperties.ApplyStyle(Style)

Description

Applies a style to the row.

Signature: C#

```
public void ApplyStyle(Style style)
```

Signature: VB.NET

```
Public Sub ApplyStyle(ByVal style As Style)
```

Parameters

style

The Style object to apply to the row.

Remarks

When a style is applied - rather than set (see RowProperties.Style) - only the differences between the new style and style properties previously assigned to the row (through the ExcelWriter API or in Microsoft Excel) will take effect. For example, if a cell in the row has a background color and the new style applied does not contain a background color, the cell's color will not be affected. However, if the new style includes a background color, it will replace the existing background color of the cell.

RowProperties.AutoFitHeight()

Description

Sets the width of the row to the height of the tallest cell value in the row.

Signature: C#

```
public void AutoFitHeight()
```

Signature: VB.NET

```
Public Sub AutoFitHeight()
```

Examples

C#

```
rowProps.AutoFitHeight();
```

vb.net

```
rowProps.AutoFitHeight()
```

RowProperties.Height

Description

Sets or returns the height of the row in points (1/72 of an inch).

Signature: C#

```
public double Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Double
```

Examples

C#

```
//--- Get Height
          double rowHeight = rowProps.Height;

          //--- Set Height
          rowProps.Height = 35;
```

vb.net

```
'--- Get Height
          Dim rowHeight As Double = rowProps.Height

          '--- Set Height
          rowProps.Height = 35
```

RowProperties.Hidden

Description

Sets or returns whether or not the row will be hidden when the workbook is opened in Excel.

Signature: C#

```
public boolean Hidden{ get; set; }
```

Signature: VB.NET

```
Public Property Hidden() As Boolean
```

Examples

C#

```
//--- Get Hidden
          bool hideRow = rowProps.Hidden;

          //--- Set Hidden
          rowProps.Hidden = false;
```

vb.net

```
'--- Get Hidden
          Dim hideRow As Boolean = rowProps.Hidden

          '--- Set Hidden
          rowProps.Hidden = False
```

RowProperties.OutlineCollapsed

Description

Sets or returns whether the row's outlined row group is collapsed. If the outline is collapsed, the row is hidden.

Signature: C#

```
public boolean OutlineCollapsed{ get; set; }
```

Signature: VB.NET

```
Public Property OutlineCollapsed() As Boolean
```

Examples

C#

```
//--- Get OutlineCollapsed
          bool collapsed = rowProps.OutlineCollapsed;

          //--- Set OutlineCollapsed
          rowProps.OutlineCollapsed = true;
```

vb.net

```
'--- Get OutlineCollapsed
          Dim collapsed As Boolean = rowProps.OutlineCollapsed

          '--- Set OutlineCollapsed
          rowProps.OutlineCollapsed = True
```

RowProperties.OutlineLevel

Description

Sets or returns the outline level of the row, as a value between 0 and 7. The outline level is used to control grouping in Excel. For example, to group several rows, set their outline levels to 1; to group a subset of rows within the group, set their outline levels to 2.

Signature: C#

```
public int OutlineLevel{ get; set; }
```

Signature: VB.NET

```
Public Property OutlineLevel() As Integer
```

Examples

C#

```
//--- Get OutlineLevel
          int level = rowProps.OutlineLevel;

          //--- Set OutlineLevel
          rowProps.OutlineLevel = 4;
```

vb.net

```
'--- Get OutlineLevel
          Dim level As Integer = rowProps.OutlineLevel

          '--- Set OutlineLevel
          rowProps.OutlineLevel = 4
```

RowProperties.RowNumber

Description

Returns the row's 0-based index.

Signature: C#

```
public int RowNumber{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property RowNumber() As Integer
```

Examples

C#

```
int rowNum = rowProps.RowNumber;
```

vb.net

```
Dim rowNum As Integer = rowProps.RowNumber
```

RowProperties.Style

Description

Sets or returns a Style object that determines how cells in the row look, including number format, font, orientation, etc. Accessing this style will not affect the formatting of other cells with the same global style.

Signature: C#

```
public Style Style{ get; set; }
```

Signature: VB.NET

```
Public Property Style() As Style
```

Remarks

When a style is set on a column - rather than applied (see ApplyStyle) - all properties of the style take effect on the cells in that row. Properties not explicitly set on the new style default to Normal style values.

Examples

C#

```
//--- Get Style
          Style rowStyle = rowProps.Style;

          //--- Set Style
          rowProps.Style = styl;
```

vb.net

```
'--- Get Style
          Dim rowStyle As Style = rowProps.Style

          '--- Set Style
          rowProps.Style = styl
```

Series

Description

A Series object represents a single data series in a chart. To return a `Series` object, use one of the SeriesCollection.CreateSeries methods.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Series
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Series
```

Properties

Name	Description
AxisType	If the chart contains two X or Y axes, AxisType sets or returns whether the series is associated with the primary or secondary axis.
BubbleSizes	Sets or returns a range formula that determines the size of the bubbles in a bubble chart.
ChartType	If the chart is a column-line chart, ChartType sets or returns the type of chart the series is associated with.
DataLabels	Returns a SeriesDataLabel object for formatting series data labels.
DataPointMarker	Returns a DataPointMarker object representing the data markers of a line, radar, or scatter series.
ErrorBarX	Returns an ErrorBar object representing the X error bars for the series. Error bars graphically express potential error amounts relative to each data marker in a data series. You can add error bars to data series in two-dimensional area, bar, column, line, scatter, and bubble charts. In scatter and bubble charts, you can display error bars for x values, y values, or both.
ErrorBarY	Returns an ErrorBar object representing the Y error bars for the series.
HasShadow	Sets or returns whether the series will be displayed with a shadow effect.
HasSmoothedLine	Sets or returns returns whether data points in a scatter series are connected by smoothed lines, that is, whether interpolated values, rather than real values, are displayed.
Interior	Returns a ChartInterior object, representing the area of the data series.
LegendEntryFont	Sets or returns the Font object for the series legend entry.

Line	Returns a ChartLine object representing either the border around the series - for most chart types - or the series line itself, for line charts.
Name	Sets or returns the name of the series. See also Series.NameFormula.
NameFormula	Sets or returns the formula containing the name of the series. See also Series.Name.
ScatterValues	Sets or returns the range of cells containing the x-values of a scatter or bubble chart data series.
SettingsBarColumn	Returns a SettingsBarColumn object representing the series properties unique to bar and column charts.
SettingsLineAreaScatter	Returns a SettingsLineAreaScatter object representing the series properties unique to line, area, and scatter charts.
SettingsPieDoughnut	Returns a SettingsPieDoughnut object representing the series properties unique to pie and doughnut charts.
ShowLegendEntry	Sets or returns whether the legend entry for this series will be shown.
Trendlines	Returns the Trendlines collection for the series. A trendline graphically shows trends in a data series, such as increased sales over a period of months.
Values	Sets or returns the range of cells containing the values of a chart data series.
VaryColorsByPoint	Sets or returns whether colors will be different for each data point in the series.

Indexers

Name	Description
Item(Int32)	Returns the DataPoint object at the specified 0-based index.

Methods

Name	Description
ChangeConfiguration(ChartType, AxisType)	Changes the ChartType and AxisType of the series at the same time. Certain combinations of charts do not allow these operations to be performed sequentially. For example, if a chart contains two pies on the primary axis and a line on the secondary, you cannot change the chart or axis type of a pie unless the changes are simultaneous.

Series.AxisType

Description

If the chart contains two X or Y axes, AxisType sets or returns whether the series is associated with the primary or secondary axis.

Signature: C#

```
public AxisType AxisType{ get; set; }
```

Signature: VB.NET

```
Public Property AxisType() As AxisType
```

Series.BubbleSizes

Description

Sets or returns a range formula that determines the size of the bubbles in a bubble chart.

Signature: C#

```
public System.String BubbleSizes{ get; set; }
```

Signature: VB.NET

```
Public Property BubbleSizes() As String
```

Remarks

Returns null if the chart is not a bubble chart.

Series.ChangeConfiguration(ChartType, AxisType)

Description

Changes the ChartType and AxisType of the series at the same time. Certain combinations of charts do not allow these operations to be performed sequentially. For example, if a chart contains two pies on the primary axis and a line on the secondary, you cannot change the chart or axis type of a pie unless the changes are simultaneous.

Signature: C#

```
public void ChangeConfiguration(ChartType type, AxisType axisType)
```

Signature: VB.NET

```
Public Sub ChangeConfiguration(ByVal type As ChartType, ByVal axisType As AxisType)
```

Parameters

type

A `ChartType` value.

axisType

An `AxisType` value.

Series.ChartType

Description

If the chart is a column-line chart, ChartType sets or returns the type of chart the series is associated with.

Signature: C#

```
public ChartType ChartType{ get; set; }
```

Signature: VB.NET

```
Public Property ChartType() As ChartType
```

Series.DataLabels

Description

Returns a SeriesDataLabel object for formatting series data labels.

Signature: C#

```
public SeriesDataLabel DataLabels{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DataLabels() As SeriesDataLabel
```

Series.DataPointMarker

Description

Returns a DataPointMarker object representing the data markers of a line, radar, or scatter series.

Signature: C#

```
public DataPointMarker DataPointMarker{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DataPointMarker() As DataPointMarker
```

Series.ErrorBarX

Description

Returns an ErrorBar object representing the X error bars for the series. Error bars graphically express potential error amounts relative to each data marker in a data series. You can add error bars to data series in two-dimensional area, bar, column, line, scatter, and bubble charts. In scatter and bubble charts, you can display error bars for x values, y values, or both.

Signature: C#

```
public ErrorBar ErrorBarX{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ErrorBarX() As ErrorBar
```

Remarks

An exception will be thrown if the current series does not support X error bars.

Series.ErrorBarY

Description

Returns an ErrorBar object representing the Y error bars for the series.

Signature: C#

```
public ErrorBar ErrorBarY{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ErrorBarY() As ErrorBar
```

Remarks

Error bars graphically express potential error amounts relative to each data marker in a data series. You can add error bars to data series in two-dimensional area, bar, column, line, scatter, and bubble charts. In scatter and bubble charts, you can display error bars for x values, y values, or both.

An exception will be thrown if the current series does not support Y error bars. The following types of series do not support Y error bars:

Series.HasShadow

Description

Sets or returns whether the series will be displayed with a shadow effect.

Signature: C#

```
public boolean HasShadow{ get; set; }
```

Signature: VB.NET

```
Public Property HasShadow() As Boolean
```

Series.HasSmoothedLine

Description

Sets or returns returns whether data points in a scatter series are connected by smoothed lines, that is, whether interpolated values, rather than real values, are displayed.

Signature: C#

```
public boolean HasSmoothedLine{ get; set; }
```

Signature: VB.NET

```
Public Property HasSmoothedLine() As Boolean
```

Series.Interior

Description

Returns a ChartInterior object, representing the area of the data series.

Signature: C#

```
public ChartInterior Interior{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Interior() As ChartInterior
```

Series.Item(Int32)

Description

Returns the DataPoint object at the specified 0-based index.

Signature: C#

```
public DataPoint this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As DataPoint
```

Parameters

index

The 0-based index of the `DataPoint` to retrieve.

Returns

The `DataPoint` object at the specified index.

Series.LegendEntryFont

Description

Sets or returns the Font object for the series legend entry.

Signature: C#

```
public Font LegendEntryFont{ get; set; }
```

Signature: VB.NET

```
Public Property LegendEntryFont() As Font
```

Series.Line

Description

Returns a ChartLine object representing either the border around the series - for most chart types - or the series line itself, for line charts.

Signature: C#

```
public ChartLine Line{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Line() As ChartLine
```

Series.Name

Description

Sets or returns the name of the series. See also Series.NameFormula.

Signature: C#

```
public System.String Name{ get; set; }
```

Signature: VB.NET

```
Public Property Name() As String
```

Remarks

If a name is not set, a default series name will be used in the worksheet. The default name format is the series number.

Series.NameFormula

Description

Sets or returns the formula containing the name of the series. See also Series.Name.

Signature: C#

```
public System.String NameFormula{ get; set; }
```

Signature: VB.NET

```
Public Property NameFormula() As String
```

Series.ScatterValues

Description

Sets or returns the range of cells containing the x-values of a scatter or bubble chart data series.

Signature: C#

```
public System.String ScatterValues{ get; set; }
```

Signature: VB.NET

```
Public Property ScatterValues() As String
```

Remarks

This property will return null for `Series` that do not represent Scatter or Bubble series, and its value (if set) will only have an effect on Scatter or Bubble series. When appropriate, the `ScatterValues` defines the location on the `ScatterValueAxis` for the series. The `ScatterValueAxis` is displayed horizontally.

Series.SettingsBarColumn

Description

Returns a SettingsBarColumn object representing the series properties unique to bar and column charts.

Signature: C#

```
public SettingsBarColumn SettingsBarColumn{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property SettingsBarColumn() As SettingsBarColumn
```

Series.SettingsLineAreaScatter

Description

Returns a SettingsLineAreaScatter object representing the series properties unique to line, area, and scatter charts.

Signature: C#

```
public SettingsLineAreaScatter SettingsLineAreaScatter{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property SettingsLineAreaScatter() As SettingsLineAreaScatter
```

Series.SettingsPieDoughnut

Description

Returns a SettingsPieDoughnut object representing the series properties unique to pie and doughnut charts.

Signature: C#

```
public SettingsPieDoughnut SettingsPieDoughnut{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property SettingsPieDoughnut() As SettingsPieDoughnut
```

Series.ShowLegendEntry

Description

Sets or returns whether the legend entry for this series will be shown.

Signature: C#

```
public boolean ShowLegendEntry{ get; set; }
```

Signature: VB.NET

```
Public Property ShowLegendEntry() As Boolean
```

Series.Trendlines

Description

Returns the Trendlines collection for the series. A trendline graphically shows trends in a data series, such as increased sales over a period of months.

Signature: C#

```
public Trendlines Trendlines{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Trendlines() As Trendlines
```

Remarks

An exception will be thrown if the current series does not support trendlines. The following types of series do not support trendlines:

Series.Values

Description

Sets or returns the range of cells containing the values of a chart data series.

Signature: C#

```
public System.String Values{ get; set; }
```

Signature: VB.NET

```
Public Property Values() As String
```

Remarks

The values of a chart data series denote the position of each data point along the primary or secondary Value Axis, as appropriate. For most charts the value axes are vertical.

Series.VaryColorsByPoint

Description

Sets or returns whether colors will be different for each data point in the series.

Signature: C#

```
public boolean VaryColorsByPoint{ get; set; }
```

Signature: VB.NET

```
Public Property VaryColorsByPoint() As Boolean
```

SeriesCollection

Description

The SeriesCollection class contains the set of data series in a worksheet.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class SeriesCollection : System.Collections.Generic.IEnumerable<Series>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class SeriesCollection
  Implements System.Collections.Generic.IEnumerable(Of Series)
```

Remarks

To return a `SeriesCollection` object, use Chart.SeriesCollection.

Properties

Name	Description
CategoryData	Sets or Returns a formula representing a set of values for the primary category (x) axis. The formula returned does not contain x axis data for a scatter series; CategoryData will return an empty string. To get scatter series data use Series.ScatterValues. This method for a bubble chart has no effect.
Count	Returns the number of data series in the collection.
SecondaryCategoryData	Sets or Returns a formula representing a set of values for the secondary category (x) axis.

Indexers

Name	Description
Item(Int32)	Returns the Series at the specified index.

Methods

Name	Description
CreateSeries(Area)	Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.
CreateSeries(Area, ChartType, AxisType)	Creates a new Series that is automatically added to the SeriesCollection.
CreateSeries(Range)	Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.
CreateSeries(Range, ChartType, AxisType)	Creates a new Series that is automatically added to the SeriesCollection.
CreateSeries(String)	Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.
CreateSeries(String, ChartType, AxisType)	Creates a new Series that is automatically added to the SeriesCollection.
GetEnumerator()	Returns an IEnumerator for the SeriesCollection of a Chart
Remove(Int32)	Removes the specified Series from the collection.
Remove(Series)	Removes the specified Series from the collection.

SeriesCollection.CategoryData

Description

Sets or Returns a formula representing a set of values for the primary category (x) axis. The formula returned does not contain x axis data for a scatter series; CategoryData will return an empty string. To get scatter series data use Series.ScatterValues. This method for a bubble chart has no effect.

Signature: C#

```
public System.String CategoryData{ get; set; }
```

Signature: VB.NET

```
Public Property CategoryData() As String
```

SeriesCollection.Count

Description

Returns the number of data series in the collection.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

SeriesCollection.CreateSeries

Overload	Description
CreateSeries(Area)	Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.

CreateSeries(Area, ChartType, AxisType)	Creates a new Series that is automatically added to the SeriesCollection.
CreateSeries(Range)	Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.
CreateSeries(Range, ChartType, AxisType)	Creates a new Series that is automatically added to the SeriesCollection.
CreateSeries(String)	Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.
CreateSeries(String, ChartType, AxisType)	Creates a new Series that is automatically added to the SeriesCollection.

SeriesCollection.CreateSeries(Area, ChartType, AxisType)

Description

Creates a new Series that is automatically added to the SeriesCollection.

Signature: C#

```
public Series CreateSeries(Area area, ChartType chartType, AxisType axisType)
```

Signature: VB.NET

```
Public Function CreateSeries(ByVal area As Area, ByVal chartType As ChartType, ByVal axisType As
AxisType) As Series
```

Parameters

area

A Area object representing the cells that contain the source data for the new series.

chartType

The type of series to add. This method will throw an exception if the specified type is not compatible with an existing series type in the chart.

axisType

Specifies whether the series will be associated with the primary or secondary axis.

Returns

A `Series` object representing the new data series.

SeriesCollection.CreateSeries(Area)

Description

Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.

Signature: C#

```
public Series CreateSeries(Area area)
```

Signature: VB.NET

```
Public Function CreateSeries(ByVal area As Area) As Series
```

Parameters

area

A Area object representing the cells that contain the source data for the new series.

Returns

A `Series` object representing the new data series.

SeriesCollection.CreateSeries(Range, ChartType, AxisType)

Description

Creates a new Series that is automatically added to the SeriesCollection.

Signature: C#

```
public Series CreateSeries(Range range, ChartType chartType, AxisType axisType)
```

Signature: VB.NET

```
Public Function CreateSeries(ByVal range As Range, ByVal chartType As ChartType, ByVal axisType As
AxisType) As Series
```

Parameters

range

A Range object representing the cells that contain the source data for the new series.

chartType

The type of series to add. This method will throw an exception if the specified type is not compatible with an existing series type in the chart.

axisType

Specifies whether the series will be associated with the primary or secondary axis.

Returns

A `Series` object representing the new data series.

SeriesCollection.CreateSeries(Range)

Description

Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.

Signature: C#

```
public Series CreateSeries(Range range)
```

Signature: VB.NET

```
Public Function CreateSeries(ByVal range As Range) As Series
```

Parameters

range

A Range object representing the cells that contain the source data for the new series.

Returns

A `Series` object representing the new data series.

SeriesCollection.CreateSeries(String, ChartType, AxisType)

Description

Creates a new Series that is automatically added to the SeriesCollection.

Signature: C#

```
public Series CreateSeries(System.String valueFormula, ChartType chartType, AxisType axisType)
```

Signature: VB.NET

```
Public Function CreateSeries(ByVal valueFormula As String, ByVal chartType As ChartType, ByVal
axisType As AxisType) As Series
```

Parameters

valueFormula

A formula representing the cells that contain the source data for the new series.

chartType

The type of series to add. The method will throw an exception if the specified type is not compatible with an existing series type in the chart.

axisType

Specifies whether the series will be associated with the primary or secondary axis.

Returns

A `Series` object representing the new data series.

SeriesCollection.CreateSeries(String)

Description

Creates a new Series that is automatically added to the SeriesCollection. The series will be plotted on the primary axis, and its type will be the chart's ChartType.

Signature: C#

```
public Series CreateSeries(System.String valueFormula)
```

Signature: VB.NET

```
Public Function CreateSeries(ByVal valueFormula As String) As Series
```

Parameters

valueFormula

A formula representing the cells that contain the source data for the new series.

Returns

A `Series` object representing the new data series.

SeriesCollection.GetEnumerator()

Description

Returns an IEnumerator for the SeriesCollection of a Chart

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the SeriesCollection of a Chart as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = chart.SeriesCollection.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = chart.SeriesCollection.GetEnumerator()
```

SeriesCollection.Item(Int32)

Description

Returns the Series at the specified index.

Signature: C#

```
public Series this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As Series
```

Parameters

index

The 0-based index of the `Series` to return.

Returns

The `Series` at the specified index.

SeriesCollection.Remove

Overload	Description
Remove(Int32)	Removes the specified Series from the collection.
Remove(Series)	Removes the specified Series from the collection.

SeriesCollection.Remove(Int32)

Description

Removes the specified Series from the collection.

Signature: C#

```
public Series Remove(int index)
```

Signature: VB.NET

```
Public Function Remove(ByVal index As Integer) As Series
```

Parameters

index

The 0-based index of the `Series` to remove.

Returns

The removed `Series`.

SeriesCollection.Remove(Series)

Description

Removes the specified Series from the collection.

Signature: C#

```
public void Remove(Series series)
```

Signature: VB.NET

```
Public Sub Remove(ByVal series As Series)
```

Parameters

series

The `Series` to remove.

SeriesCollection.SecondaryCategoryData

Description

Sets or Returns a formula representing a set of values for the secondary category (x) axis.

Signature: C#

```
public System.String SecondaryCategoryData{ get; set; }
```

Signature: VB.NET

```
Public Property SecondaryCategoryData() As String
```

SeriesDataLabel

Description

A SeriesDataLabel represents the appearance of series labels.

Signature: C#

```
public sealed class SeriesDataLabel : ChartText
```

Signature: VB.NET

```
Public NotInheritable Class SeriesDataLabel
  Inherits ChartText
```

Remarks

To returns a `SeriesDataLabel` object, use Series.DataLabels.

Properties

Name	Description
ContainsBubbleSize	Sets or returns whether the data label will contain the bubble size. This will only be displayed in bubble charts.
ContainsCategoryName	Sets or returns whether series data labels will contain the category name (x value).
ContainsSeriesName	Sets or returns whether the data labels will contain the series name.
ContainsValue	Sets or returns whether the data label will contain the actual (y) value for the data point.
ContainsValueAsPercentage	Sets or returns whether the data label will contain the (y) value for the data point as a percentage. Only doughnut and pie charts can dispay values as percentages.
DataLabelPlacement	Sets or returns where the data label should be placed relative to its data point
LabelItemSeperator	Sets or returns the string seperating the different elements in the data label.
ShowLegendKey	Sets or returns whether the data label will contain the legend key for the series.

SeriesDataLabel.ContainsBubbleSize

Description

Sets or returns whether the data label will contain the bubble size. This will only be displayed in bubble charts.

Signature: C#

```
public boolean ContainsBubbleSize{ get; set; }
```

Signature: VB.NET

```
Public Property ContainsBubbleSize() As Boolean
```

SeriesDataLabel.ContainsCategoryName

Description

Sets or returns whether series data labels will contain the category name (x value).

Signature: C#

```
public boolean ContainsCategoryName{ get; set; }
```

Signature: VB.NET

```
Public Property ContainsCategoryName() As Boolean
```

SeriesDataLabel.ContainsSeriesName

Description

Sets or returns whether the data labels will contain the series name.

Signature: C#

```
public boolean ContainsSeriesName{ get; set; }
```

Signature: VB.NET

```
Public Property ContainsSeriesName() As Boolean
```

SeriesDataLabel.ContainsValue

Description

Sets or returns whether the data label will contain the actual (y) value for the data point.

Signature: C#

```
public boolean ContainsValue{ get; set; }
```

Signature: VB.NET

```
Public Property ContainsValue() As Boolean
```

SeriesDataLabel.ContainsValueAsPercentage

Description

Sets or returns whether the data label will contain the (y) value for the data point as a percentage. Only doughnut and pie charts can dispay values as percentages.

Signature: C#

```
public boolean ContainsValueAsPercentage{ get; set; }
```

Signature: VB.NET

```
Public Property ContainsValueAsPercentage() As Boolean
```

SeriesDataLabel.DataLabelPlacement

Introduced in build 4.5.1

Description

Sets or returns where the data label should be placed relative to its data point

Signature: C#

```
public DataLabelPlacementType DataLabelPlacement{ get; set; }
```

Signature: VB.NET

```
Public Property DataLabelPlacement() As DataLabelPlacementType
```

Remarks

Set the position of each SeriesDataLabel using the DataLabelPlacementType enum.

Examples

C#

```
//--- Create chart
Chart chart = ws.Charts.CreateChart(ChartType.Column.Stacked, ws.CreateAnchor(10, 0, 0, 0));

//--- Create data series
Series series = chart.SeriesCollection.CreateSeries("A1:A10");

//--- Set the data label position for the first SeriesDataLabel to OutsideEnd
series[0].Label.DataLabelPlacement = DataLabelPlacementType.OutsideEnd;
```

vb.net

```
'--- Create chart
Dim chart As Chart = ws.Charts.CreateChart(ChartType.Column.Stacked, ws.CreateAnchor(10, 0, 0, 0))

'--- Create data series
Dim series As Series = chart.SeriesCollection.CreateSeries("A1:A10")

'--- Set the data label position for the first SeriesDataLabel to OutsideEnd
series(0).Label.DataLabelPlacement = DataLabelPlacementType.OutsideEnd
```

SeriesDataLabel.LabelItemSeperator

Description

Sets or returns the string seperating the different elements in the data label.

Signature: C#

```
public System.String LabelItemSeperator{ get; set; }
```

Signature: VB.NET

```
Public Property LabelItemSeperator() As String
```

SeriesDataLabel.ShowLegendKey

Description

Sets or returns whether the data label will contain the legend key for the series.

Signature: C#

```
public boolean ShowLegendKey{ get; set; }
```

Signature: VB.NET

```
Public Property ShowLegendKey() As Boolean
```

SettingsBarColumn

Description

The SettingsBarColumn class contains all settings that are specific to bar and column charts. To return a `SettingsBarColumn` object,

Signature: C#

```
public sealed class SettingsBarColumn
```

Signature: VB.NET

```
Public NotInheritable Class SettingsBarColumn
```

Properties

Name	Description
DataShapeType	Sets or returns the shape type used for three-dimensional charts.
Overlap	Sets or returns the overlap between bars or columns for a category. The value will must be between -100 and 100. The higher the value, the greater the overlap within the category.
SpaceBetweenCategories	Sets or returns the space between categories in bar and column charts.

Nested Classes

Name	Description
ShapeType	SettingsBarColumn.ShapeType class contains the shapes that can be used for three-dimensional column and bar charts.

SettingsBarColumn.DataShapeType

Description

Sets or returns the shape type used for three-dimensional charts.

Signature: C#

```
public ShapeType DataShapeType{ get; set; }
```

Signature: VB.NET

```
Public Property DataShapeType() As ShapeType
```

SettingsBarColumn.Overlap

Description

Sets or returns the overlap between bars or columns for a category. The value will must be between -100 and 100. The higher the value, the greater the overlap within the category.

Signature: C#

```
public int Overlap{ get; set; }
```

Signature: VB.NET

```
Public Property Overlap() As Integer
```

SettingsBarColumn.ShapeType

Description

SettingsBarColumn.ShapeType class contains the shapes that can be used for three-dimensional column and bar charts.

Signature: C#

```
public sealed class ShapeType : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ShapeType
  Inherits TypesafeEnum
```

Values

Name	Description
Box	Box
Cone	Cone
CutCone	CutCone
CutPyramid	CutPyramid
Cylinder	Cylinder
Pyramid	Pyramid

SettingsBarColumn.SpaceBetweenCategories

Description

Sets or returns the space between categories in bar and column charts.

Signature: C#

```
public int SpaceBetweenCategories{ get; set; }
```

Signature: VB.NET

```
Public Property SpaceBetweenCategories() As Integer
```

SettingsLineAreaScatter

Description

The SettingsLineAreaScatter class contains all settings that are specific to line, area, and scatter charts. To return a `SettingsLineAreaScatter` object,

Signature: C#

```
public sealed class SettingsLineAreaScatter
```

Signature: VB.NET

```
Public NotInheritable Class SettingsLineAreaScatter
```

Properties

Name	Description
DropLine	Returns a ChartLine object representing the drop lines of a line, area, or scatter series.

HighLowLine	Returns a ChartLine object representing the high-low lines of a line or scatter series.
ShowDropLines	Sets or returns whether series drop-lines will be displayed in the chart.
ShowHighLowLines	Sets or returns whether series high-low lines will be displayed in the chart.
ShowUpDownBars	Sets or returns whether series up/down bars will be displayed in the chart.
UpDownBar	Returns an UpDownBar object representing the up/down bars of a line series.

SettingsLineAreaScatter.DropLine

Description

Returns a ChartLine object representing the drop lines of a line, area, or scatter series.

Signature: C#

```
public ChartLine DropLine{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DropLine() As ChartLine
```

SettingsLineAreaScatter.HighLowLine

Description

Returns a ChartLine object representing the high-low lines of a line or scatter series.

Signature: C#

```
public ChartLine HighLowLine{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property HighLowLine() As ChartLine
```

SettingsLineAreaScatter.ShowDropLines

Description

Sets or returns whether series drop-lines will be displayed in the chart.

Signature: C#

```
public boolean ShowDropLines{ get; set; }
```

Signature: VB.NET

```
Public Property ShowDropLines() As Boolean
```

SettingsLineAreaScatter.ShowHighLowLines

Description

Sets or returns whether series high-low lines will be displayed in the chart.

Signature: C#

```
public boolean ShowHighLowLines{ get; set; }
```

Signature: VB.NET

```
Public Property ShowHighLowLines() As Boolean
```

SettingsLineAreaScatter.ShowUpDownBars

Description

Sets or returns whether series up/down bars will be displayed in the chart.

Signature: C#

```
public boolean ShowUpDownBars{ get; set; }
```

Signature: VB.NET

```
Public Property ShowUpDownBars() As Boolean
```

SettingsLineAreaScatter.UpDownBar

Description

Returns an UpDownBar object representing the up/down bars of a line series.

Signature: C#

```
public UpDownBar UpDownBar{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property UpDownBar() As UpDownBar
```

SettingsPieDoughnut

Description

The SettingsPieDoughnut class contains all settings that are specific to pie and doughnut charts. To return a `SettingsPieDoughnut` object,

Signature: C#

```
public sealed class SettingsPieDoughnut
```

Signature: VB.NET

```
Public NotInheritable Class SettingsPieDoughnut
```

Properties

Name	Description
AngleOfFirstSlice	Sets or returns the angle of the first slice in the pie or doughnut chart.
BopPopGapSize	Sets or returns the size of the gap between pies in a pie of pie chart or between bar and pie in a bar of pie chart.

BopPopSecondChartSize	Sets or returns the size of the second chart in a pie of pie or bar of pie chart, as a percentage of the size of the first chart.
BopPopSeriesLines	Returns a ChartLine object representing the series lines from the main pie chart to the secondary chart in a bar of pie or pie of pie chart.
BopPopSplitPercentage	Sets or returns a percentage that determines which of a chart's data points will be included in the bar of a bar of pie chart, or second pie of a pie of pie chart.
BopPopSplitPosition	Sets or returns a number of positions that determines which of a chart's data points will be included in the bar of a bar of pie chart, or second pie of a pie of pie chart.
BopPopSplitType	Sets or returns how data will be split in the bar of a bar of pie chart, or second pie of a pie of pie chart.
BopPopSplitValue	Sets or returns a value that determines which of a chart's data points will be included in the bar of a bar of pie chart, or second pie of a pie of pie chart. The data points included in the second chart are those with values less than the specified value. The value only takes effect if the chart's BopPopSplitType is set to value.
LeaderLines	Returns a ChartLine object representing the leader lines in the pie chart.
ShowLeaderLines	Sets or returns whether lines from the slices to their data labels will be shown.
SizeOfCenterHole	Sets or returns the size of the center hole of a doughnut chart as a percentage of the doughnut size.
SlicesDistanceFromCenter	Sets or returns the distance - in an exploded pie chart - of slices from the chart's center.
UseDefaultSplitValue	Sets or returns whether the two parts of the bar of pie or pie of pie chart will be split using the default value, which is determined by Excel.

Nested Classes

Name	Description
BopPopSplit	A BobPopSplitValue determines what type of values will be included in the bar of a bar of pie chart, or second pie of a pie of pie chart.

SettingsPieDoughnut.AngleOfFirstSlice

Description

Sets or returns the angle of the first slice in the pie or doughnut chart.

Signature: C#

```
public int AngleOfFirstSlice{ get; set; }
```

Signature: VB.NET

```
Public Property AngleOfFirstSlice() As Integer
```

SettingsPieDoughnut.BopPopGapSize

Description

Sets or returns the size of the gap between pies in a pie of pie chart or between bar and pie in a bar of pie chart.

Signature: C#

```
public int BopPopGapSize{ get; set; }
```

Signature: VB.NET

```
Public Property BopPopGapSize() As Integer
```

SettingsPieDoughnut.BopPopSecondChartSize

Description

Sets or returns the size of the second chart in a pie of pie or bar of pie chart, as a percentage of the size of the first chart.

Signature: C#

```
public int BopPopSecondChartSize{ get; set; }
```

Signature: VB.NET

```
Public Property BopPopSecondChartSize() As Integer
```

SettingsPieDoughnut.BopPopSeriesLines

Description

Returns a ChartLine object representing the series lines from the main pie chart to the secondary chart in a bar of pie or pie of pie chart.

Signature: C#

```
public ChartLine BopPopSeriesLines{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property BopPopSeriesLines() As ChartLine
```

SettingsPieDoughnut.BopPopSplit

Description

A BobPopSplitValue determines what type of values will be included in the bar of a bar of pie chart, or second pie of a pie of pie chart.

Signature: C#

```
public sealed class BopPopSplit : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class BopPopSplit
  Inherits TypesafeEnum
```

Values

Name	Description
Custom	Defaults to BopPopSplit.Position
Percent	The bar or second pie will include data points with percentages less than the percentage specified by BopPopSplitPercentage.
Position	The bar or second pie will contain the first chart's last *n* data points, where *n* is the number specified by BopPopSplitPosition.
Value	The bar or second pie will include data points with values less than the value specified by BopPopSplitValue.

SettingsPieDoughnut.BopPopSplitPercentage

Description

Sets or returns a percentage that determines which of a chart's data points will be included in the bar of a bar of pie chart, or second pie of a pie of pie chart.

Signature: C#

```
public int BopPopSplitPercentage{ get; set; }
```

Signature: VB.NET

```
Public Property BopPopSplitPercentage() As Integer
```

Remarks

The data points included in the second chart are those with percentages less than the specified percentage. The percentage only takes effect if the chart's BopPopSplitType is set to Percent.

SettingsPieDoughnut.BopPopSplitPosition

Description

Sets or returns a number of positions that determines which of a chart's data points will be included in the bar of a bar of pie chart, or second pie of a pie of pie chart.

Signature: C#

```
public int BopPopSplitPosition{ get; set; }
```

Signature: VB.NET

```
Public Property BopPopSplitPosition() As Integer
```

Remarks

The bar or second pie will contain the chart's last *n* data points, where *n* is the number of positions set. The position only takes effect if the chart's BopPopSplitType is set to Position.

SettingsPieDoughnut.BopPopSplitType

Description

Sets or returns how data will be split in the bar of a bar of pie chart, or second pie of a pie of pie chart.

Signature: C#

```
public BopPopSplit BopPopSplitType{ get; set; }
```

Signature: VB.NET

```
Public Property BopPopSplitType() As BopPopSplit
```

SettingsPieDoughnut.BopPopSplitValue

Description

Sets or returns a value that determines which of a chart's data points will be included in the bar of a bar of pie chart, or second pie of a pie of pie chart. The data points included in the second chart are those with values less than the specified value. The value only takes effect if the chart's BopPopSplitType is set to value.

Signature: C#

```
public double BopPopSplitValue{ get; set; }
```

Signature: VB.NET

```
Public Property BopPopSplitValue() As Double
```

SettingsPieDoughnut.LeaderLines

Description

Returns a ChartLine object representing the leader lines in the pie chart.

Signature: C#

```
public ChartLine LeaderLines{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property LeaderLines() As ChartLine
```

SettingsPieDoughnut.ShowLeaderLines

Description

Sets or returns whether lines from the slices to their data labels will be shown.

Signature: C#

```
public boolean ShowLeaderLines{ get; set; }
```

Signature: VB.NET

```
Public Property ShowLeaderLines() As Boolean
```

SettingsPieDoughnut.SizeOfCenterHole

Description

Sets or returns the size of the center hole of a doughnut chart as a percentage of the doughnut size.

Signature: C#

```
public int SizeOfCenterHole{ get; set; }
```

Signature: VB.NET

```
Public Property SizeOfCenterHole() As Integer
```

SettingsPieDoughnut.SlicesDistanceFromCenter

Description

Sets or returns the distance - in an exploded pie chart - of slices from the chart's center.

Signature: C#

```
public int SlicesDistanceFromCenter{ get; set; }
```

Signature: VB.NET

```
Public Property SlicesDistanceFromCenter() As Integer
```

SettingsPieDoughnut.UseDefaultSplitValue

Description

Sets or returns whether the two parts of the bar of pie or pie of pie chart will be split using the default value, which is determined by Excel.

Signature: C#

```
public boolean UseDefaultSplitValue{ get; set; }
```

Signature: VB.NET

```
Public Property UseDefaultSplitValue() As Boolean
```

Shape

Description

The Shape class allows the appearance and behavior of drawing object to be manipulated. On its own, `Shape` represents a simple auto shape, such as a line, rectangle, or oval. A shape may also represent the display area of a Comment or the size, position, and formatting of a Picture.

Signature: C#

```
public class Shape
```

Signature: VB.NET

```
Public Class Shape
```

Remarks

Once a `Shape` is created with a particular ShapeType, the type may not be changed. Instead, delete the shape and create a new one with the appropriate type.

To create a `Shape` object, call Shapes.CreateShape.

Properties

Name	Description
FillColor	Sets or returns the shape's fill color. The color may be either a custom color or a palette color.
FillTransparency	Sets or returns the fill transparency the shape as a real number value between 0 (opaque) and 1 (transparent).
FitToText	Sets or returns whether the shape will be automatically sized to fit the text.
Height	Sets or returns the height of the Shape.
HorizontalAlignment	Sets or returns the horizontal alignment of text within the shape.
LineColor	Sets or returns the Shape 's line color.
LockAspectRatio	Sets or returns whether the aspect ratio of the shape will be locked in Excel. This setting has no effect within ExcelWriter.
Name	Returns the name associated with this shape.
ObjectPositioning	Sets or returns the ObjectPositioning value that determines the shape's resize and reposition behavior when the cells in the worksheet are changed.

Position	Sets or returns the Shape 's position in the worksheet.
Rotation	Sets or returns the rotation angle of the shape.
Text	Sets or returns the text displayed within the shape.
Type	Returns the Shape object's type. This property may not be changed.
VerticalAlignment	Sets or returns the vertical alignment of text within the shape.
Width	Sets or returns the width of the Shape.

Methods

Name	Description
ClearRichTextFormatting()	Clears all rich text formatting within the shape. The font of the shape text will be set to the font set (through the ExcelWriter API) for the cell the shape is attached to.
GetCharacters(Int32)	Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.
GetCharacters(Int32, Int32)	Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.
SetCustomFillColor(Int32, Int32, Int32)	Sets the fill color for the shape to be the specified custom RGB value, and returns the generated `Color`. This method allows a shape to have a color not defined in the worksheet's Palette.
SetCustomLineColor(Int32, Int32, Int32)	Sets the line color for the shape to be the specified custom RGB value, and returns the generated `Color`. This method allows a shape to have a line color not defined in the worksheet's Palette.

Shape.ClearRichTextFormatting()

Description

Clears all rich text formatting within the shape. The font of the shape text will be set to the font set (through the ExcelWriter API) for the cell the shape is attached to.

Signature: C#

```
public virtual void ClearRichTextFormatting()
```

Signature: VB.NET

```
Public Overridable Sub ClearRichTextFormatting()
```

Shape.FillColor

Description

Sets or returns the shape's fill color. The color may be either a custom color or a palette color.

Signature: C#

```
public Color FillColor{ get; set; }
```

Signature: VB.NET

```
Public Property FillColor() As Color
```

Shape.FillTransparency

Description

Sets or returns the fill transparency the shape as a real number value between 0 (opaque) and 1 (transparent).

Signature: C#

```
public double FillTransparency{ get; set; }
```

Signature: VB.NET

```
Public Property FillTransparency() As Double
```

Shape.FitToText

Description

Sets or returns whether the shape will be automatically sized to fit the text.

Signature: C#

```
public boolean FitToText{ get; set; }
```

Signature: VB.NET

```
Public Property FitToText() As Boolean
```

Shape.GetCharacters

Overload	Description
GetCharacters(Int32)	Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.
GetCharacters(Int32, Int32)	Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.

Shape.GetCharacters(Int32, Int32)

Description

Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.

Signature: C#

```
public virtual CharacterRun GetCharacters(int startIndex, int length)
```

Signature: VB.NET

```
Public Overridable Function GetCharacters(ByVal startIndex As Integer, ByVal length As Integer) As
CharacterRun
```

Parameters

startIndex

The character position at which to begin the character run.

length

The length of the character run.

Returns

A `CharacterRun` object representing a range of characters within the cell.

Shape.GetCharacters(Int32)

Description

Returns a CharacterRun object representing a range of characters within the comment text. Use this to set the font on specified characters within the text.

Signature: C#

```
public virtual CharacterRun GetCharacters(int startIndex)
```

Signature: VB.NET

```
Public Overridable Function GetCharacters(ByVal startIndex As Integer) As CharacterRun
```

Parameters

startIndex

The 0-based character position at which to begin the character run.

Returns

A `CharacterRun` object representing a range of characters within the comment text.

Shape.Height

Description

Sets or returns the height of the Shape.

Signature: C#

```
public double Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Double
```

Remarks

If the `Shape` is directly within a Worksheet, the height is in points (1/72 inch). If the `Shape` is within Chartsheet, the height is a percentage of the chart's height (0-100).

If you set a shape's height (or Shape.Width) to a value read from a spreadsheet, the shape may lose precision due to the way Excel stores shape dimensions.

Shape.HorizontalAlignment

Description

Sets or returns the horizontal alignment of text within the shape.

Signature: C#

```
public HAlign HorizontalAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property HorizontalAlignment() As HAlign
```

Shape.LineColor

Description

Sets or returns the Shape 's line color.

Signature: C#

```
public Color LineColor{ get; set; }
```

Signature: VB.NET

```
Public Property LineColor() As Color
```

Shape.LockAspectRatio

Description

Sets or returns whether the aspect ratio of the shape will be locked in Excel. This setting has no effect within ExcelWriter.

Signature: C#

```
public boolean LockAspectRatio{ get; set; }
```

Signature: VB.NET

```
Public Property LockAspectRatio() As Boolean
```

Shape.Name

Description

Returns the name associated with this shape.

Signature: C#

```
public System.String Name{ get; set; }
```

Signature: VB.NET

```
Public Property Name() As String
```

Shape.ObjectPositioning

Description

Sets or returns the ObjectPositioning value that determines the shape's resize and reposition behavior when the cells in the worksheet are changed.

Signature: C#

```
public ObjectPositioning ObjectPositioning{ get; set; }
```

Signature: VB.NET

```
Public Property ObjectPositioning() As ObjectPositioning
```

Shape.Position

Description

Sets or returns the Shape 's position in the worksheet.

Signature: C#

```
public Anchor Position{ get; set; }
```

Signature: VB.NET

```
Public Property Position() As Anchor
```

Shape.Rotation

Description

Sets or returns the rotation angle of the shape.

Signature: C#

```
public double Rotation{ get; set; }
```

Signature: VB.NET

```
Public Property Rotation() As Double
```

Shape.SetCustomFillColor(Int32, Int32, Int32)

Description

Sets the fill color for the shape to be the specified custom RGB value, and returns the generated `Color`. This method allows a shape to have a color not defined in the worksheet's Palette.

Signature: C#

```
public virtual Color SetCustomFillColor(int red, int green, int blue)
```

Signature: VB.NET

```
Public Overridable Function SetCustomFillColor(ByVal red As Integer, ByVal green As Integer, ByVal
blue As Integer) As Color
```

Parameters

red

The red component of the RGB value.

green

The green component of the RGB value.

blue

The blue component of the RGB value.

Returns

A `Color` object representing the fill color.

Shape.SetCustomLineColor(Int32, Int32, Int32)

Description

Sets the line color for the shape to be the specified custom RGB value, and returns the generated `Color`. This method allows a shape to have a line color not defined in the worksheet's Palette.

Signature: C#

```
public virtual Color SetCustomLineColor(int red, int green, int blue)
```

Signature: VB.NET

```
Public Overridable Function SetCustomLineColor(ByVal red As Integer, ByVal green As Integer, ByVal
blue As Integer) As Color
```

Parameters

red

The red component of the RGB value.

green

The green component of the RGB value.

blue

The blue component of the RGB value.

Returns

A `Color` object representing the line color.

Shape.Text

Description

Sets or returns the text displayed within the shape.

Signature: C#

```
public System.String Text{ get; set; }
```

Signature: VB.NET

```
Public Property Text() As String
```

Shape.Type

Description

Returns the Shape object's type. This property may not be changed.

Signature: C#

```
public ShapeType Type{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Type() As ShapeType
```

Shape.VerticalAlignment

Description

Sets or returns the vertical alignment of text within the shape.

Signature: C#

```
public VAlign VerticalAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property VerticalAlignment() As VAlign
```

Shape.Width

Description

Sets or returns the width of the Shape.

Signature: C#

```
public double Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Double
```

Remarks

If the `Shape` is directly within a Worksheet, the width is in points (1/72 inch). If the `Shape` is within Chartsheet, the width is a percentage of the chart's width (0-100).

If you set a shape's width (or Shape.Height) to a value read from a spreadsheet, the shape may lose precision due to the way Excel stores shape dimensions.

Shapes

Description

The Shapes collection contains all Shape objects in the worksheet. To return a `Shapes` collection, use Worksheet.Shapes.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Shapes : System.Collections.Generic.IEnumerable<Shape>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Shapes
  Implements System.Collections.Generic.IEnumerable(Of Shape)
```

Properties

Name	Description
Count	Returns the number of Shape objects in the collection.

Indexers

Name	Description
Item(Int32)	Returns the Shape object at the specified position in the Shapes collection.
Item(String)	Returns the Shape object with the specified name in the Shapes collection.

Methods

Name	Description
AddShape(Shape)	Adds the specified Shape object to the Shapes collection.
CreateShape(ShapeType, Anchor)	Creates a shape with the specified position and type. The specified shape type must not be Picture, Group, Comment, or Unknown. The new shape is automatically added to the Shapes collection.
GetEnumerator()	Returns an IEnumerator for the Shapes in a Worksheet
ListShapes()	Lists all the Shape object in the Shapes collection. This does not return the underlying collection, so changes to the returned value will have no effect on future calls to this `Shapes` object.
Remove(Int32)	Removes the Shape object at the specified position from the collection. The position will correspond to the position of the `Shape` within the array returned by ListShapes.
Remove(Shape)	Removes the specified Shape object from the collection. If the specified `Shape` object does not exist in the collection, no change is made.

Shapes.AddShape(Shape)

Description

Adds the specified Shape object to the Shapes collection.

Signature: C#

```
public void AddShape(Shape shape)
```

Signature: VB.NET

```
Public Sub AddShape(ByVal shape As Shape)
```

Parameters

shape

The `Shape` object to add to the collection.

Shapes.Count

Description

Returns the number of Shape objects in the collection.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Shapes.CreateShape(ShapeType, Anchor)

Description

Creates a shape with the specified position and type. The specified shape type must not be Picture, Group, Comment, or Unknown. The new shape is automatically added to the Shapes collection.

Signature: C#

```
public Shape CreateShape(ShapeType type, Anchor aPosition)
```

Signature: VB.NET

```
Public Function CreateShape(ByVal type As ShapeType, ByVal aPosition As Anchor) As Shape
```

Parameters

type

A ShapeType value representing the type of shape to create.

aPosition

An Anchor object representing the position of the new shape in the worksheet.

Returns

A Shape object representing the new shape.

Shapes.GetEnumerator()

Description

Returns an IEnumerator for the Shapes in a Worksheet

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the Shapes in a Worksheet as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = ws.Shapes.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = ws.Shapes.GetEnumerator()
```

Shapes.Item

Overload	Description
Item(Int32)	Returns the Shape object at the specified position in the Shapes collection.
Item(String)	Returns the Shape object with the specified name in the Shapes collection.

Shapes.Item(Int32)

Description

Returns the Shape object at the specified position in the Shapes collection.

Signature: C#

```
public Shape this[int position] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal position As Integer) As Shape
```

Parameters

position

A valid index between 0 and Shapes.Count.

Returns

The `Shape` object at the specified position.

Shapes.Item(String)

Description

Returns the Shape object with the specified name in the Shapes collection.

Signature: C#

```
public Shape this[System.String name] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal name As String) As Shape
```

Parameters

name

The name of the `Shape` to return.

Returns

The `Shape` object with the specified name.

Shapes.ListShapes()

Description

Lists all the Shape object in the Shapes collection. This does not return the underlying collection, so changes to the returned value will have no effect on future calls to this `Shapes` object.

Signature: C#

```
public Shape[] ListShapes()
```

Signature: VB.NET

```
Public Function ListShapes() As Shape()
```

Returns

An array of all the `Shape` objects in the collection.

Shapes.Remove

Overload	Description
Remove(Int32)	Removes the Shape object at the specified position from the collection. The position will correspond to the position of the `Shape` within the array returned by ListShapes.
Remove(Shape)	Removes the specified Shape object from the collection. If the specified `Shape` object does not exist in the collection, no change is made.

Shapes.Remove(Int32)

Description

Removes the Shape object at the specified position from the collection. The position will correspond to the position of the `Shape` within the array returned by ListShapes.

Signature: C#

```
public void Remove(int position)
```

Signature: VB.NET

```
Public Sub Remove(ByVal position As Integer)
```

Parameters

position

The 0-based position of the `Shape` to remove.

Shapes.Remove(Shape)

Description

Removes the specified Shape object from the collection. If the specified `Shape` object does not exist in the collection, no change is made.

Signature: C#

```
public void Remove(Shape shape)
```

Signature: VB.NET

```
Public Sub Remove(ByVal shape As Shape)
```

Parameters

shape

The `Shape` object to remove from the collection.

ShapeType

Description

The ShapeType class contains the Excel shape types supported by ExcelWriter.

Signature: C#

```
public sealed class ShapeType
```

Signature: VB.NET

```
Public NotInheritable Class ShapeType
```

Values

Name	Description
Arc	Arc
BentArrow	BentArrow
BentUpArrow	BentUpArrow
Bevel	Bevel
BlockArc	Block arc
BracePair	Brace pair
BracketPair	Bracket pair
Can	Can
Cube	Cube
Chart	Chart
Chart2	Chart2
Chevron	Chevron
CircularArrow	CircularArrow
ComboBox	ComboBox
Comment	Comment
CurvedDownArrow	CurvedDownArrow
CurvedLeftArrow	CurvedLeftArrow
CurvedRightArrow	CurvedRightArrow
CurvedUpArrow	CurvedUpArrow
Diamond	Diamond
Donut	Donut
DoubleWave	DoubleWave
DownArrow	DownArrow
DownArrowCallout	DownArrowCallout
DownRibbon	DownRibbon
Ellipse	Ellipse
EllipseDownRibbon	EllipseDownRibbon
EllipseUpRibbon	EllipseUpRibbon
Explosion1	Explosion1
Explosion2	Explosion2
FlowChartCollate	FlowChartCollate
FlowChartConnector	FlowChartConnector

FlowChartDecision	FlowChartDecision
FlowChartDelay	FlowChartDelay
FlowChartDisplay	FlowChartDisplay
FlowChartDocument	FlowChartDocument
FlowChartExtract	FlowChartExtract
FlowChartInputOutput	FlowChartInputOutput
FlowChartInternalStorage	FlowChartInternalStorage
FlowChartMagneticDisk	FlowChartMagneticDisk
FlowChartMagneticDrum	FlowChartMagneticDrum
FlowChartMagneticTape	FlowChartMagneticTape
FlowChartManualInput	FlowChartManualInput
FlowChartManualOperation	FlowChartManualOperation
FlowChartMerge	FlowChartMerge
FlowChartMultidocument	FlowChartMultidocument
FlowChartOfflineStorage	FlowChartOfflineStorage
FlowChartOnlineStorage	FlowChartOnlineStorage
FlowChartOr	FlowChartOr
FlowChartPredefinedProcess	FlowChartPredefinedProcess
FlowChartPreparation	FlowChartPreparation
FlowChartProcess	FlowChartProcess
FlowChartPunchedCard	FlowChartPunchedCard
FlowChartPunchedTape	FlowChartPunchedTape
FlowChartSort	FlowChartSort
FlowChartSummingJunction	FlowChartSummingJunction
FlowChartTerminator	FlowChartTerminator
FoldedCorner	Folded corner
Group	Group
Heart	Heart
Hexagon	Hexagon
HorizontalScroll	HorizontalScroll
IsocelesTriangle	Isoceles triangle
LeftArrow	LeftArrow
LeftArrowCallout	LeftArrowCallout
LeftBrace	Left brace
LeftBracket	Left bracket
LeftRightArrow	LeftRightArrow
LeftRightArrowCallout	LeftRightArrowCallout
LeftRightUpArrow	LeftRightUpArrow
LeftUpArrow	LeftUpArrow

LightningBolt	Lightning bolt
Line	Line
Moon	Moon
NotchedRightArrow	NotchedRightArrow
Octagon	Octagon
Parallelogram	Parallelogram
Pentagon	Pentagon
PentagonArrow	PentagonArrow
Picture	Picture
Plaque	Plaque
Plus	Plus
Prohibited	Prohibited
QuadArrow	QuadArrow
QuadArrowCallout	QuadArrowCallout
Rectangle	Rectangle
RightArrow	RightArrow
RightArrowCallout	RightArrowCallout
RightBrace	Right brace
RightBracket	Right bracket
RightTriangle	Right triangle
RoundRectangle	Round rectangle
SmileyFace	Smiley face
Star4Point	Star4Point
Star8Point	Star8Point
Star16Point	Star16Point
Star24Point	Star24Point
Star32Point	Star32Point
StripedRightArrow	StripedRightArrow
Sun	Sun
TextBox	TextBox
Trapezoid	Trapezoid
Unsupported	Unsupported
UpArrow	UpArrow
UpArrowCallout	UpArrowCallout
UpDownArrow	UpDownArrow
UpDownArrowCallout	UpDownArrowCallout
UpRibbon	UpRibbon
UTurnArrow	UTurnArrow
VerticalScroll	VerticalScroll

Wave	Wave

Style

Description

ExcelWriter has three style types: CellStyle, GlobalStyle, and NamedStyle.

Signature: C#

```
public class Style
```

Signature: VB.NET

```
Public Class Style
```

Remarks

All three types derive from the Style class, and `NamedStyle` derives from `GlobalStyle`.

You cannot create a CellStyle, only manipulate existing ones as part of the objects they apply to. To create a `GlobalStyle`, use Workbook.CreateStyle(). To create a `NamedStyle`, use Workbook.CreateNamedStyle(). To get a `NamedStyle` from an existing document, use Workbook.GetNamedStyle().

Examples

C#

```
//--- Open existing spreadsheet
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Open(@"C:\MySpreadsheet.xls");

          //--- Create GlobalStyle
          Style global = wb.CreateStyle();

          //--- Create NamedStyle
          Style newNamed = wb.CreateNamedStyle("MyStyle");

          //--- Get NamedStyle
          Style existingNamed = wb.GetNamedStyle("ExistingStyle");
```

vb.net

```
'--- Open existing spreadsheet
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Open("C:\MySpreadsheet")

          '--- Create GlobalStyle
          Dim global As Style = wb.CreateStyle()

          '--- Create NamedStyle
          Dim newNamed As Style = wb.CreateNamedStyle("MyStyle")

          '--- Get NamedStyle
          Dim existingNamed As Style = wb.GetNamedStyle("ExistingStyle")
```

Properties

Name	Description
BackgroundColor	Sets or returns the background color for cells.

Border	Returns a Border object for the style.
CellLocked	Sets or returns whether a cell will be locked in a protected worksheet.
Font	Sets or returns the Font object that defines the font for this style.
ForegroundColor	Sets or returns a foreground color for cells. A foreground color is visible when a pattern is assigned to a cell.
HideFormulas	Sets or returns whether formulas or their calculated results will be displayed in a protected worksheet.
HorizontalAlignment	Sets or returns the horizontal alignment of text within cells.
IndentLevel	Sets or returns level of indentation of the contents within the cell.
JustifyDistributed	Sets or returns the whether the text will be justfied horizontally, when the Horizontal Alignment is set to Distributed
NumberFormat	Sets or returns the number format string used for formatting of numbers and dates.
Orientation	Sets or returns the angle that text is displayed at, as a value between -90 and 90.
Pattern	Sets or returns a pattern applied to cells.
ShrinkToFit	Sets or returns whether text in a cell should be reduced in size to fit within the cell's width.
TextDirection	Sets or returns the direction of the text with in the cells.
VerticalAlignment	Sets or returns the vertical alignment of text within cells.
WrapText	Sets or returns if text that is larger than the width of the cell will be wrapped, increasing the height of a cell.

Nested Classes

Name	Description
ColumnInsertBehavior	A ColumnInsertBehavior value determines what style will be applied to the inserted columns.
HAlign	The HAlign class contains all horizontal alignment options.
RowInsertBehavior	A RowInsertBehavior value determines what style will be applied to the inserted rows.
TextDir	The TextDirection class contains all text direction options.
VAlign	The VAlign class contains all vertical alignment options.

Style.BackgroundColor

Description

Sets or returns the background color for cells.

Signature: C#

```
public Color BackgroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property BackgroundColor() As Color
```

Examples

C#

```
//--- Get BackgroungColor
          Color bgColor = styl.BackgroundColor;

          //--- Set BackgroundColor
          styl.BackgroundColor = wb.Palette.GetClosestColor(162, 221, 139);
```

vb.net

```
'--- Get BackgroundColor
          Dim bgColor As Color = styl.BackgroundColor

          '--- Set BackgroundColor
          styl.BackgroundColor = wb.Palette.GetClosestColor(162, 221, 139)
```

Style.Border

Description

Returns a Border object for the style.

Signature: C#

```
public Border Border{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Border() As Border
```

Examples

C#

```
Border bdr = styl.Border;
```

vb.net

```
Dim bdr As Border = styl.Border
```

Style.CellLocked

Description

Sets or returns whether a cell will be locked in a protected worksheet.

Signature: C#

```
public boolean CellLocked{ get; set; }
```

Signature: VB.NET

```
Public Property CellLocked() As Boolean
```

Examples

C#

```
//--- Get CellLocked
          bool locked = style.CellLocked;

          //--- Set CellLocked
          style.CellLocked = true;
```

vb.net

```
'--- Get CellLocked
          Dim locked As Boolean = style.CellLocked

          '--- Set CellLocked
          style.CellLocked = True
```

Style.ColumnInsertBehavior

Description

A ColumnInsertBehavior value determines what style will be applied to the inserted columns.

Signature: C#

```
public sealed class ColumnInsertBehavior : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class ColumnInsertBehavior
  Inherits TypesafeEnum
```

Values

Name	Description
Ignore	The default style will be applied to the new columns.
Left	The style of the new columns will match the style of the column to their left
Right	The style of the new columns will match the style of the column to their right

Style.Font

Description

Sets or returns the Font object that defines the font for this style.

Signature: C#

```
public Font Font{ get; set; }
```

Signature: VB.NET

```
Public Property Font() As Font
```

Examples

C#

```
//--- Get Font
          Font fnt = styl.Font;

          //--- Set Font
          styl.Font = fnt;
```

vb.net

```
'--- Get Font
          Dim fnt As Font = styl.Font

          '--- Set Font
          styl.Font = fnt
```

Style.ForegroundColor

Description

Sets or returns a foreground color for cells. A foreground color is visible when a pattern is assigned to a cell.

Signature: C#

```
public Color ForegroundColor{ get; set; }
```

Signature: VB.NET

```
Public Property ForegroundColor() As Color
```

Examples

C#

```
//--- Get ForegroundColor
          Color fgColor = styl.ForegroundColor;

          //--- Set ForegroundColor
          styl.ForegroundColor = wb.Palette.GetClosestColor(162, 221, 139);
```

vb.net

```
'--- Get ForegroundColor
          Dim fgColor As Color = styl.ForegroundColor

          '--- Set ForegroundColor
          styl.ForegroundColor = wb.Palette.GetClosestColor(162, 221, 139)
```

Style.HAlign

Description

The HAlign class contains all horizontal alignment options.

Signature: C#

```
public sealed class HAlign : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class HAlign
  Inherits TypesafeEnum
```

Values

Name	Description
Center	Text will be centered horizontally.
CenterAcrossSelection	Center across selection.
Distributed	Text will be distributed horizontally.
Fill	Fill
General	General
Justify	Text will be justified.
Left	Text will be left-aligned.
Right	Text will be right-aligned.

Style.HideFormulas

Description

Sets or returns whether formulas or their calculated results will be displayed in a protected worksheet.

Signature: C#

```
public boolean HideFormulas{ get; set; }
```

Signature: VB.NET

```
Public Property HideFormulas() As Boolean
```

Examples

C#

```
//--- Get HideFormulas
          bool hide = styl.HideFormulas;

          //--- Set HideFormulas
          styl.HideFormulas = true;
```

vb.net

```
'--- Get HideFormulas
          Dim hide As Boolean = styl.HideFormulas

          '--- Set HideFormulas
          styl.HideFormulas = True
```

Style.HorizontalAlignment

Description

Sets or returns the horizontal alignment of text within cells.

Signature: C#

```
public HAlign HorizontalAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property HorizontalAlignment() As HAlign
```

Examples

C#

```
//--- Get HorizontalAlignment
          Style.HAlign ha = styl.HorizontalAlignment;

          //--- Set HorizontalAlignment
          styl.HorizontalAlignment = Style.HAlign.Center;
```

vb.net

```
'--- Get HorizontalAlignment
          Dim ha As Style.HAlign = styl.HorizontalAlignment

          '--- Set HorizontalAlignment
          styl.HorizontalAlignment = Style.HAlign.Center
```

Style.IndentLevel

Description

Sets or returns level of indentation of the contents within the cell.

Signature: C#

```
public int IndentLevel{ get; set; }
```

Signature: VB.NET

```
Public Property IndentLevel() As Integer
```

Examples

C#

```
//--- Get IndentLevel
          int indentLevel = style.IndentLevel;

          //--- Set IndentLevel
          style.IndentLevel= 4;
```

vb.net

```
'--- Get IndentLevel
          Dim indentLevel As Integer = style.IndentLevel

          '--- Set IndentLevel
          style.IndentLevel = 2
```

Style.JustifyDistributed

Description

Sets or returns the whether the text will be justfied horizontally, when the Horizontal Alignment is set to Distributed

Signature: C#

```
public boolean JustifyDistributed{ get; set; }
```

Signature: VB.NET

```
Public Property JustifyDistributed() As Boolean
```

Examples

C#

```
//--- Get Justify Distributed
          bool justifyDist = styl.JustifyDistributed;

          //--- Set Justify Distributed
          styl.JustifyDistributed = true;
```

vb.net

```
'--- Get Justify Distributed
          Dim justifyDist As Boolean = styl.JustifyDistributed

          '--- Set Justify Distributed
          styl.JustifyDistributed = True
```

Style.NumberFormat

Description

Sets or returns the number format string used for formatting of numbers and dates.

Signature: C#

```
public System.String NumberFormat{ get; set; }
```

Signature: VB.NET

```
Public Property NumberFormat() As String
```

Examples

C#

```
//--- Get NumberFormat
          string numFormat = styl.NumberFormat;

          //--- Set NumberFormat
          styl.NumberFormat = "#.##";

          //--- Set NumberFormat
          styl.NumberFormat = "mm/dd/yyyy hh:mm:ss AM/PM";
```

vb.net

```
'--- Get NumberFormat
          Dim numFormat As String = styl.NumberFormat

          '--- Set NumberFormat
          styl.NumberFormat = "#.##"

          '--- Set NumberFormat
          styl.NumberFormat = "mm/dd/yyyy hh:mm:ss AM/PM"
```

Style.Orientation

Description

Sets or returns the angle that text is displayed at, as a value between -90 and 90.

Signature: C#

```
public int Orientation{ get; set; }
```

Signature: VB.NET

```
Public Property Orientation() As Integer
```

Examples

C#

```
//--- Get Orientation
          int orient = styl.Orientation;

          //--- Set Orientation
          styl.Orientation = 45;
```

vb.net

```
'--- Get Orientation
          Dim orient As Integer = styl.Orientation

          '--- Set Orientation
          styl.Orientation = 45
```

Style.Pattern

Description

Sets or returns a pattern applied to cells.

Signature: C#

```
public Pattern Pattern{ get; set; }
```

Signature: VB.NET

```
Public Property Pattern() As Pattern
```

Examples

C#

```
//--- Get Pattern
          Pattern ptn = c.Style.Pattern;

          //--- Set Pattern
          styl.Pattern = Pattern.DiagonalCrosshatch;
```

vb.net

```
'--- Get Pattern
          Dim ptn As Pattern = c.Style.Pattern

          '--- Set Pattern
          styl.Pattern = Patter.DiagonalCrosshatch
```

Style.RowInsertBehavior

Description

A RowInsertBehavior value determines what style will be applied to the inserted rows.

Signature: C#

```
public sealed class RowInsertBehavior : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class RowInsertBehavior
  Inherits TypesafeEnum
```

Values

Name	Description
Above	The style of the new rows will match the style of the row above them.
Below	The style of the new rows will match the style of the row below them.
Ignore	The default style will be applied to the new rows.

Style.ShrinkToFit

Description

Sets or returns whether text in a cell should be reduced in size to fit within the cell's width.

Signature: C#

```
public boolean ShrinkToFit{ get; set; }
```

Signature: VB.NET

```
Public Property ShrinkToFit() As Boolean
```

Examples

C#

```
//--- Get ShrinkToFit
          bool shrink = styl.ShrinkToFit;

          //--- Set ShrinkToFit
          styl.ShrinkToFit = true;
```

vb.net

```
'--- Get ShrinkToFit
          Dim shrink As Boolean = styl.ShrinkToFit

          '--- Set ShrinkToFit
          styl.ShrinkToFit = True
```

Style.TextDir

Description

The TextDirection class contains all text direction options.

Signature: C#

```
public sealed class TextDir : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class TextDir
  Inherits TypesafeEnum
```

Values

Name	Description
Context	Text is displayed in the context of the version of Excel that opens the spreadsheet.
LeftToRight	Text is displayed left to right.
RightToLeft	Text is displayed right to left.

Style.TextDirection

Description

Sets or returns the direction of the text with in the cells.

Signature: C#

```
public TextDir TextDirection{ get; set; }
```

Signature: VB.NET

```
Public Property TextDirection() As TextDir
```

Examples

C#

```
//--- Get TextDirection
          Style.TextDir textDir = styl.TextDirection;

          //--- Set TextDirection
          styl.TextDirection = Styl.TextDir.RightToLeft;
```

vb.net

```
'--- Get TextDirection
          Dim textDir As Style.TextDir = styl.TextDirection

          '--- Set TextDirection
          styl.TextDirection = Style.TextDir.RightToLeft
```

Style.VAlign

Description

The VAlign class contains all vertical alignment options.

Signature: C#

```
public sealed class VAlign : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class VAlign
  Inherits TypesafeEnum
```

Values

Name	Description
Bottom	Text will be bottom-aligned.
Center	Text will be centered vertically.
Distributed	Text will be distributed vertically.
Justify	Text will be justified vertically.
Top	Text will be top-aligned.

Style.VerticalAlignment

Description

Sets or returns the vertical alignment of text within cells.

Signature: C#

```
public VAlign VerticalAlignment{ get; set; }
```

Signature: VB.NET

```
Public Property VerticalAlignment() As VAlign
```

Examples

C#

```
//--- Get VerticalAlignment
          Style.VAlign vert = styl.VerticalAlignment;

          //--- Set VerticalAlignment
          styl.VerticalAlignment = Style.VAlign.Bottom;
```

vb.net

```
'--- Get VerticalAlignment
          Dim vert As Style.VAlign = styl.VerticalAlignment

          '--- Set VerticalAlignment
          styl.VerticalAlignment = Style.VAlign.Bottom
```

Style.WrapText

Description

Sets or returns if text that is larger than the width of the cell will be wrapped, increasing the height of a cell.

Signature: C#

```
public boolean WrapText{ get; set; }
```

Signature: VB.NET

```
Public Property WrapText() As Boolean
```

Examples

C#

```
//--- Get WrapText
          bool wrap = styl.WrapText;

          //--- Set WrapText
          styl.WrapText = true;
```

vb.net

```
'--- Get WrapText
          Dim wrap As Boolean = styl.WrapText

          '--- Set WrapText
          styl.WrapText = True
```

Trendline

Description

A Trendline object represents a series trendline. To create a `Trendline` object, call Trendlines.CreateTrendline .

Signature: C#

```
public sealed class Trendline
```

Signature: VB.NET

```
Public NotInheritable Class Trendline
```

Properties

Name	Description
BackcastPeriods	For all trendline types except moving average, sets or returns the number of periods the regression should extrapolate backward.
ForecastPeriods	For all trendline types except moving average, sets or returns the number of periods the regression should extrapolate forward.
Intercept	For all trendline types except moving average, sets or returns the point on the value axis where the trendline should intercept.
LegendEntryFont	Sets or returns the Font object for the trendline's legend entry.
Line	Returns a ChartLine object representing the appearance of the trendline.
Name	Sets or returns the name of the trendline.
RegressionType	Sets or returns the type of regression used to construct this trendline.
RegressionValue	Sets the or returns the value used to generate the regression equation.
ShowEquation	Sets or returns whether the label will contain the equation for the trendline.
ShowLegendEntry	Sets or returns whether an entry for the trendline will be displayed in the chart legend.
ShowRSquaredValue	Sets or returns whether the label will contain the R-squared value for the Trendline's equation.
TrendlineLabel	Returns a ChartText object representing a trendline label.
UseIntercept	Controls whether the intercept value will be used by the trendline.

Methods

Name	Description
ClearIntercept()	Specifies that the intercept value should not be used.

Nested Classes

Name	Description
Regression	A Trendline.Regression value specifies type of trendline to display on the graph.

Trendline.BackcastPeriods

Description

For all trendline types except moving average, sets or returns the number of periods the regression should extrapolate backward.

Signature: C#

```
public double BackcastPeriods{ get; set; }
```

Signature: VB.NET

```
Public Property BackcastPeriods() As Double
```

Trendline.ClearIntercept()

Description

Specifies that the intercept value should not be used.

Signature: C#

```
public void ClearIntercept()
```

Signature: VB.NET

```
Public Sub ClearIntercept()
```

Trendline.ForecastPeriods

Description

For all trendline types except moving average, sets or returns the number of periods the regression should extrapolate forward.

Signature: C#

```
public double ForecastPeriods{ get; set; }
```

Signature: VB.NET

```
Public Property ForecastPeriods() As Double
```

Trendline.Intercept

Description

For all trendline types except moving average, sets or returns the point on the value axis where the trendline should intercept.

Signature: C#

```
public double Intercept{ get; set; }
```

Signature: VB.NET

```
Public Property Intercept() As Double
```

Trendline.LegendEntryFont

Description

Sets or returns the Font object for the trendline's legend entry.

Signature: C#

```
public Font LegendEntryFont{ get; set; }
```

Signature: VB.NET

```
Public Property LegendEntryFont() As Font
```

Trendline.Line

Description

Returns a ChartLine object representing the appearance of the trendline.

Signature: C#

```
public ChartLine Line{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Line() As ChartLine
```

Trendline.Name

Description

Sets or returns the name of the trendline.

Signature: C#

```
public System.String Name{ get; set; }
```

Signature: VB.NET

```
Public Property Name() As String
```

Trendline.Regression

Description

A Trendline.Regression value specifies type of trendline to display on the graph.

Signature: C#

```
public sealed class Regression : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class Regression
  Inherits TypesafeEnum
```

Values

Name	Description
Exponential	A curved line that is most useful when data values rise or fall at increasingly higher rates. You cannot create an exponential trendline if your data contains zero or negative values.
Logarithmic	A best-fit curved line used when the rate of change in the data increases or decreases quickly and then levels out.
MovingAverage	A line that smoothes out fluctuations in data to show a pattern or trend more clearly.
Polynomial	A curved line used when data fluctuates. The order of the polynomial can be determined by the number of fluctuations in the data or by how many bends (hills and valleys) appear in the curve.
Power	A curved line that is best used with data sets that compare measurements that increase at a specific rate.

Trendline.RegressionType

Description

Sets or returns the type of regression used to construct this trendline.

Signature: C#

```
public Regression RegressionType{ get; set; }
```

Signature: VB.NET

```
Public Property RegressionType() As Regression
```

Trendline.RegressionValue

Description

Sets the or returns the value used to generate the regression equation.

Signature: C#

```
public int RegressionValue{ get; set; }
```

Signature: VB.NET

```
Public Property RegressionValue() As Integer
```

Remarks

For Polynomial regression trendlines, the value will be the order of the polynomial. For MovingAverage trendlines, the value specifies the number of periods over which the average is computed.

An exception will be thrown if the following conditions are not met:

Trendline.ShowEquation

Description

Sets or returns whether the label will contain the equation for the trendline.

Signature: C#

```
public boolean ShowEquation{ get; set; }
```

Signature: VB.NET

```
Public Property ShowEquation() As Boolean
```

Trendline.ShowLegendEntry

Description

Sets or returns whether an entry for the trendline will be displayed in the chart legend.

Signature: C#

```
public boolean ShowLegendEntry{ get; set; }
```

Signature: VB.NET

```
Public Property ShowLegendEntry() As Boolean
```

Trendline.ShowRSquaredValue

Description

Sets or returns whether the label will contain the R-squared value for the Trendline's equation.

Signature: C#

```
public boolean ShowRSquaredValue{ get; set; }
```

Signature: VB.NET

```
Public Property ShowRSquaredValue() As Boolean
```

Trendline.TrendlineLabel

Description

Returns a ChartText object representing a trendline label.

Signature: C#

```
public ChartText TrendlineLabel{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property TrendlineLabel() As ChartText
```

Trendline.UseIntercept

Description

Controls whether the intercept value will be used by the trendline.

Signature: C#

```
public boolean UseIntercept{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property UseIntercept() As Boolean
```

Trendlines

Description

Trendlines is a collection of all Trendline objects associated with a data series in a chart. To return a `Trendlines` collection, use Series.Trendlines .

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Trendlines : System.Collections.Generic.IEnumerable<Trendline>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Trendlines
  Implements System.Collections.Generic.IEnumerable(Of Trendline)
```

Properties

Name	Description
Count	Returns the number of Trendline objects in the Trendlines collection.

Indexers

Name	Description
Item(Int32)	Returns the Trendline object at the specified 0-based index.

Methods

Name	Description
CreateTrendline(Regression)	Creates a new Trendline object that is automatically added to the Trendlines collection.
GetEnumerator()	Returns an IEnumerator for the Trendlines in a Series
Remove(Int32)	Removes the Trendline at the specified index from the collection.
Remove(Trendline)	Removes the specified Trendline from the collection.

Trendlines.Count

Description

Returns the number of Trendline objects in the Trendlines collection.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Trendlines.CreateTrendline(Trendline.Regression)

Description

Creates a new Trendline object that is automatically added to the Trendlines collection.

Signature: C#

```
public Trendline CreateTrendline(Regression type)
```

Signature: VB.NET

```
Public Function CreateTrendline(ByVal type As Regression) As Trendline
```

Parameters

type

One of the following Trendline.Regression : Exponential , Logarithmic , MovingAverage , Polynomial , Power .

Returns

A `Trendline` object representing a new trendline.

Trendlines.GetEnumerator()

Description

Returns an IEnumerator for the Trendlines in a Series

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the Trendlines in a Series as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = series.Trendlines.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = series.Trendlines.GetEnumerator()
```

Trendlines.Item(Int32)

Description

Returns the Trendline object at the specified 0-based index.

Signature: C#

```
public Trendline this[int index] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal index As Integer) As Trendline
```

Parameters

index

The 0-based index of the `Trendline` to return.

Returns

The `Trendline` object at the specified index.

Trendlines.Remove

Overload	Description
Remove(Int32)	Removes the Trendline at the specified index from the collection.

Remove(Trendline)	Removes the specified Trendline from the collection.

Trendlines.Remove(Int32)

Description

Removes the Trendline at the specified index from the collection.

Signature: C#

```
public Trendline Remove(int index)
```

Signature: VB.NET

```
Public Function Remove(ByVal index As Integer) As Trendline
```

Parameters

index

The 0-based index of the `Trendline` to remove.

Returns

The removed `Trendline` object.

Trendlines.Remove(Trendline)

Description

Removes the specified Trendline from the collection.

Signature: C#

```
public void Remove(Trendline trendline)
```

Signature: VB.NET

```
Public Sub Remove(ByVal trendline As Trendline)
```

Parameters

trendline

The `Trendline` to remove.

UpDownBar

Description

An UpDownBar object represents up/down bars in a line chart. Up/down bars connect data points in the highest and lowest series in a line chart to show motion during an interval.

Signature: C#

```
public sealed class UpDownBar
```

Signature: VB.NET

```
Public NotInheritable Class UpDownBar
```

Remarks

To return an `UpDownBar` object, use SettingsLineAreaScatter.UpDownBar .

Properties

Name	Description
DownBarBorderLine	Returns a ChartLine object representing the border around the down bar portion of the
DownBarInterior	Returns a ChartInterior object representing the area of the down bar portion of the UpDownBar object.
GapBetweenBars	Sets or returns the width of the gap between the up/down bars.
UpBarBorderLine	Returns a ChartLine object representing the border around the up bar portion of the
UpBarInterior	Returns a ChartInterior object representing the area of the up bar portion of the UpDownBar object.

UpDownBar.DownBarBorderLine

Description

Returns a ChartLine object representing the border around the down bar portion of the

Signature: C#

```
public ChartLine DownBarBorderLine{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DownBarBorderLine() As ChartLine
```

UpDownBar.DownBarInterior

Description

Returns a ChartInterior object representing the area of the down bar portion of the UpDownBar object.

Signature: C#

```
public ChartInterior DownBarInterior{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DownBarInterior() As ChartInterior
```

UpDownBar.GapBetweenBars

Description

Sets or returns the width of the gap between the up/down bars.

Signature: C#

```
public int GapBetweenBars{ get; set; }
```

Signature: VB.NET

```
Public Property GapBetweenBars() As Integer
```

UpDownBar.UpBarBorderLine

Description

Returns a ChartLine object representing the border around the up bar portion of the

Signature: C#

```
public ChartLine UpBarBorderLine{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property UpBarBorderLine() As ChartLine
```

UpDownBar.UpBarInterior

Description

Returns a ChartInterior object representing the area of the up bar portion of the UpDownBar object.

Signature: C#

```
public ChartInterior UpBarInterior{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property UpBarInterior() As ChartInterior
```

ValueAxis

Description

A ValueAxis object represents a chart's value (y) axis. The `ValueAxis` class extends Axis .

Signature: C#

```
public sealed class ValueAxis : Axis
```

Signature: VB.NET

```
Public NotInheritable Class ValueAxis
  Inherits Axis
```

Properties

Name	Description
CategoryAxisCrossMaxValue	Sets or returns whether the category axis will cross the value axis at its maximum value.
CategoryCrossingValue	Sets or returns the Y value at which the category axis crosses the value axis.
MajorIncrement	Sets or returns the major increment of the axis, the increment between its major gridlines and/or major ticks marks.
MaxValue	Sets or returns the maximum value on the y axis.
MinorIncrement	Sets or returns the minor unit of measurement of the axis, the gap between its minor gridlines and/or minor ticks marks.
MinValue	Sets or returns the minimum value on the y axis.

UseLogarithmicScale	Sets or returns whether a logarithmic scale should be used on the axis.

ValueAxis.CategoryAxisCrossMaxValue

Description

Sets or returns whether the category axis will cross the value axis at its maximum value.

Signature: C#

```
public boolean CategoryAxisCrossMaxValue{ get; set; }
```

Signature: VB.NET

```
Public Property CategoryAxisCrossMaxValue() As Boolean
```

ValueAxis.CategoryCrossingValue

Description

Sets or returns the Y value at which the category axis crosses the value axis.

Signature: C#

```
public double CategoryCrossingValue{ get; set; }
```

Signature: VB.NET

```
Public Property CategoryCrossingValue() As Double
```

ValueAxis.MajorIncrement

Description

Sets or returns the major increment of the axis, the increment between its major gridlines and/or major ticks marks.

Signature: C#

```
public double MajorIncrement{ get; set; }
```

Signature: VB.NET

```
Public Property MajorIncrement() As Double
```

Remarks

If you set this to a negative value, it will set the major increment of the axis to auto. The value -1 will be returned if the major increment is set to auto.

ValueAxis.MaxValue

Description

Sets or returns the maximum value on the y axis.

Signature: C#

```
public double MaxValue{ get; set; }
```

Signature: VB.NET

```
Public Property MaxValue() As Double
```

ValueAxis.MinorIncrement

Description

Sets or returns the minor unit of measurement of the axis, the gap between its minor gridlines and/or minor ticks marks.

Signature: C#

```
public double MinorIncrement{ get; set; }
```

Signature: VB.NET

```
Public Property MinorIncrement() As Double
```

Remarks

If you set this to a negative value, it will set the minor increment of the axis to auto. The value -1 will be returned if the minor increment is set to auto.

ValueAxis.MinValue

Description

Sets or returns the minimum value on the y axis.

Signature: C#

```
public double MinValue{ get; set; }
```

Signature: VB.NET

```
Public Property MinValue() As Double
```

ValueAxis.UseLogarithmicScale

Description

Sets or returns whether a logarithmic scale should be used on the axis.

Signature: C#

```
public boolean UseLogarithmicScale{ get; set; }
```

Signature: VB.NET

```
Public Property UseLogarithmicScale() As Boolean
```

Workbook

Description

A Workbook object represents an Excel workbook. To return a `Workbook` object: Call ExcelApplication.Create() to create a new Excel workbook. Call ExcelApplication.Open() to open an existing Excel workbook.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Workbook : System.Collections.Generic.IEnumerable<Worksheet>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Workbook
  Implements System.Collections.Generic.IEnumerable(Of Worksheet)
```

Examples

C#

```
//--- Create a new Workbook
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();

          //--- Open an existing Workbook
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Open(@"C:\Reports\Report.xls");
```

vb.net

```
'--- Create a new Workbook
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()

          '--- Open an existing Workbook
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Open("C:\Reports\Report.xls")
```

Properties

Name	Description
DisplayTabs	Shows or hides the worksheet tabs at the bottom of the workbook window.
DocumentProperties	Returns a DocumentProperties object that represents properties of the specified workbook, such as name, author, and summary. In Microsoft Excel, these properties are displayed in the file's Properties sheet. To view the Properties sheet in Excel, open Excel's File menu, and select Properties.
EncryptPassword	Sets or returns the password used to encrypt the spreadsheet. Standard "Office 97/2000 Compatible" encryption is used to encrypt the spreadsheet. This is the default encryption used by Excel 97 and later versions.
ExcludeMacros	Sets or returns whether ExcelWriter should remove macros from the generated workbook or leave them in.
FirstDisplayedTab	Sets or returns the first worksheet tab to display in the tab bar at the bottom of the workbook window.
Height	Sets or returns the height of the workbook's window
HorizontalPosition	Sets or returns the horizontal (x) position of the workbook window's upper left corner when displayed in Excel.
IsProtected	Returns whether or not the worksheet is protected.
NamedRanges	Returns the named ranges of the Workbook as an array of NamedObject objects as an `IEnumerable` collection. Though the named ranges are not returned in any particular order, they iterable.
NumberFormat	Returns the NumberFormat used for formatting of numbers and dates.
Palette	Returns a Palette object that represents the specified workbook's color palette.

PasswordToModify	This sets the workbook so that it can be opened in read-only mode with the option to open in read-write mode, if the user supplies a matching password when opening the spreadsheet.
ProtectPasswordHash	Sets or returns the password hash that is used to protect the workbook.
ReadOnly	Determines whether the workbook can only be opened in read-only mode.
ReadOnlyRecommended	Sets or returns whether Excel will recommend - when the workbook is opened - that it be viewed in read-only mode.
ShowHorizontalScrollBar	When true, the horizontal scroll bar will be shown in the generated workbook. When false, the horizontal scroll bar will be hidden.
ShowVerticalScrollBar	Shows or hides the vertical scroll bar in the generated workbook.
TabRatio	Sets or returns the ratio between the workbook's tab bar and the horizontal scroll bar.
UseRCFormulaNotation	Indicates or sets the type of references used in this workbook.
VerticalPosition	Sets or returns the vertical (y) position of the workbook window's upper left corner when displayed in Excel.
Width	Sets or returns the width of the workbook's window.
Worksheets	Returns a Worksheets collection which provides access to all worksheets in the specified workbook, including charts displayed as sheets.

Indexers

Name	Description
Item(Int32)	Returns the worksheet specified by a 0-based index. This property is an indexer for the Workbook class.
Item(String)	Returns the worksheet with the specified name. This property is an indexer for the Workbook class.

Methods

Name	Description
CreateConditionalFormat()	Creates a new ConditionalFormat object that can be set on Cells, Areas and Ranges.
CreateCopyPasteProperties(CopyPasteType)	Creates a CopyPasteProperties object with the specified data type properties set.
CreateDataImportProperties()	Creates a DataImportProperties object that contains a set of properties that can be used when importing data to a Worksheet or Area.
CreateDataValidation(ValidationType, ComparisonType, Object)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, ComparisonType, Object, Object)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, ComparisonType, String)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, ComparisonType, String, String)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, Object)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, String)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateNamedRange(Area(), String)	Creates a Range of cells from an array of Areas. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.
CreateNamedRange(String, String)	Creates a named Range of cells from a specified formula. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.
CreateNamedStyle(String)	Creates a named global style, which can be bound to cells, areas, ranges, rows, and columns. Any later changes to this style will affect all cells to which the style was assigned.
CreateRange(String)	Creates a Range of cells from a specified formula. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.

CreateRange(Area())	Creates a Range of cells from an array of Areas. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.
CreateStyle()	Creates a global style, which can be bound to cells, areas, ranges, rows, and columns. Any later changes to this style will affect all cells to which the style was bound.
GetEnumerator()	Returns an IEnumerator for the Worksheets in a Workbook
GetNamedObject(String)	Returns the NamedObject object that represents the name of a specified array, number, picture, or range. If the named object does not exist, the method returns null.
GetNamedRange(String)	Returns the specified named Range. If the named range does not exist, the method returns null.
GetNamedStyle(String)	Returns a NamedStyle if it exists. If the style does not exist, null is returned.
GetTextHeight(String, Font, Double)	Returns the minimum height required to fit the given text string using the given font.
GetTextWidth(String, Font, Double)	Returns the minimum width required to fit the given text string using the given font.
Protect(String)	Protects the workbook structure using the supplied password. End users will not be able to add or remove worksheets unless they enter the password.
Unprotect()	Removes protection from the workbook.

Workbook.CreateConditionalFormat()

Description

Creates a new ConditionalFormat object that can be set on Cells, Areas and Ranges.

Signature: C#

```
public ConditionalFormat CreateConditionalFormat()
```

Signature: VB.NET

```
Public Function CreateConditionalFormat() As ConditionalFormat
```

Returns

A `ConditionalFormat` object representing a conditional format that Excel will apply to a cell or group of cells if a specified condition is met.

Remarks

A Cell, Area or Range object holds its own copy of the ConditionalFormat when set; changes made to a `ConditionalFormat` after it is set will not affect the copies.

Examples

C#

```
wb.CreateConditionalFormat();
```

vb.net

```
wb.CreateConditionalFormat()
```

Workbook.CreateCopyPasteProperties(CopyPasteProperties.CopyPasteType)

Description

Creates a CopyPasteProperties object with the specified data type properties set.

Signature: C#

```
public CopyPasteProperties CreateCopyPasteProperties(CopyPasteType typeMask)
```

Signature: VB.NET

```
Public Function CreateCopyPasteProperties(ByVal typeMask As CopyPasteType) As CopyPasteProperties
```

Parameters

typeMask

A CopyPasteProperties constant value indicating the types of data to copy from an external worksheet.

Returns

A `CopyPasteProperties` object.

Remarks

This object is used when copying data from an external worksheet.

Used with the Worksheet.CopyPaste method of the Worksheet.

Examples

C#

```
CopyPasteProperties properties =
               ws.CreateCopyPasteProperties(
               CopyPasteProperties.ValuesOnly);
```

vb.net

```
Dim properties As CopyPasteProperties = _
               ws.CreateCopyPasteProperties( _
               CopyPasteProperties.ValuesOnly)
```

Workbook.CreateDataImportProperties()

Description

Creates a DataImportProperties object that contains a set of properties that can be used when importing data to a Worksheet or Area.

Signature: C#

```
public DataImportProperties CreateDataImportProperties()
```

Signature: VB.NET

```
Public Function CreateDataImportProperties() As DataImportProperties
```

Returns

A DataImportProperties object.

Examples

C#

```
DataImportProperties importProps =
                wb.CreateDataImportProperties();
```

vb.net

```
Dim importProps As DataImportProperties = _
                wb.CreateDataImportProperties()
```

Workbook.CreateDataValidation

Overload	Description
CreateDataValidation(ValidationType, ComparisonType, Object)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, ComparisonType, Object, Object)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, ComparisonType, String)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, ComparisonType, String, String)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, Object)	Creates a DataValidation object which can be set on a Range, Area, or Cell.
CreateDataValidation(ValidationType, String)	Creates a DataValidation object which can be set on a Range, Area, or Cell.

Workbook.CreateDataValidation(DataValidation.ValidationType, DataValidation.ComparisonType, Object, Object)

Description

Creates a DataValidation object which can be set on a Range, Area, or Cell.

Signature: C#

```
public DataValidation CreateDataValidation(ValidationType allowedType, ComparisonType comparison,
System.Object minimumValue, System.Object maximumValue)
```

Signature: VB.NET

```
Public Function CreateDataValidation(ByVal allowedType As ValidationType, ByVal comparison As
ComparisonType, ByVal minimumValue As Object, ByVal maximumValue As Object) As DataValidation
```

Parameters

allowedType

The data type allowed in the cell.

comparison

The comparison used to validate the cell's value.

minimumValue

A primitive or Date representing cell's maximum value.

maximumValue

A primitive or Date representing the cell's minimum value.

Returns

A DataValidation object.

Examples

C#

```
DataValidation dataValidation =
               wb.createDataValidation(
               DataValidation.ValidationType.WholeNumber,
               DataValidation.ComparisonType.Between, 1, 100);
```

vb.net

```
Dim dataValidation As DataValidation = _
               wb.createDataValidation( _
               DataValidation.ValidationType.WholeNumber, _
               DataValidation.ComparisonType.Between, 1, 100)
```

Workbook.CreateDataValidation(DataValidation.ValidationType, DataValidation.ComparisonType, Object)

Description

Creates a DataValidation object which can be set on a Range, Area, or Cell.

Signature: C#

```
public DataValidation CreateDataValidation(ValidationType allowedType, ComparisonType comparison,
System.Object value)
```

Signature: VB.NET

```
Public Function CreateDataValidation(ByVal allowedType As ValidationType, ByVal comparison As
ComparisonType, ByVal value As Object) As DataValidation
```

Parameters

allowedType

The data type allowed in the cell.

comparison

The comparison used to validate the cell's value.

value

A primitive or Date representing the constraint value for the data validation.

Returns

A DataValidation object.

Examples

C#

```
DataValidation dataValidation =
               wb.createDataValidation(
               DataValidation.ValidationType.WholeNumber,
               DataValidation.ComparisonType.LessThan, 1);
```

vb.net

```
Dim dataValidation As DataValidation = _
                wb.createDataValidation( _
                DataValidation.ValidationType.WholeNumber, _
                DataValidation.ComparisonType.LessThan, 1)
```

Workbook.CreateDataValidation(DataValidation.ValidationType, DataValidation.ComparisonType, String, String)

Description

Creates a DataValidation object which can be set on a Range, Area, or Cell.

Signature: C#

```
public DataValidation CreateDataValidation(ValidationType allowedType, ComparisonType comparison,
System.String minimumValue, System.String maximumValue)
```

Signature: VB.NET

```
Public Function CreateDataValidation(ByVal allowedType As ValidationType, ByVal comparison As
ComparisonType, ByVal minimumValue As String, ByVal maximumValue As String) As DataValidation
```

Parameters

allowedType

The data type allowed in the cell.

comparison

The comparison used to validate the cell's value.

minimumValue

A formula representing the cell's maximum value.

maximumValue

A formula representing the cell's minimum value.

Returns

A DataValidation object.

Examples

C#

```
DataValidation dataValidation =
                wb.createDataValidation(
                DataValidation.ValidationType.WholeNumber,
                DataValidation.ComparisonType.Between, "=1", "=100");
```

vb.net

```
Dim dataValidation As DataValidation = _
                wb.createDataValidation( _
                DataValidation.ValidationType.WholeNumber, _
                DataValidation.ComparisonType.Between, "=1", "=100")
```

Workbook.CreateDataValidation(DataValidation.ValidationType, DataValidation.ComparisonType, String)

Description

Creates a DataValidation object which can be set on a Range, Area, or Cell.

Signature: C#

```
public DataValidation CreateDataValidation(ValidationType allowedType, ComparisonType comparison,
System.String value)
```

Signature: VB.NET

```
Public Function CreateDataValidation(ByVal allowedType As ValidationType, ByVal comparison As
ComparisonType, ByVal value As String) As DataValidation
```

Parameters

allowedType

The data type allowed in the cell.

comparison

The comparison used to validate the cell's value.

value

A formula representing the constraint for the data validation.

Returns

A DataValidation object.

Examples

C#

```
DataValidation dataValidation =
               wb.createDataValidation(
               DataValidation.ValidationType.WholeNumber,
               DataValidation.ComparisonType.LessThan, "=1");
```

vb.net

```
Dim dataValidation As DataValidation = _
               wb.createDataValidation( _
               DataValidation.ValidationType.WholeNumber, _
               DataValidation.ComparisonType.LessThan, "=1")
```

Workbook.CreateDataValidation(DataValidation.ValidationType, Object)

Description

Creates a DataValidation object which can be set on a Range, Area, or Cell.

Signature: C#

```
public DataValidation CreateDataValidation(ValidationType allowedType, System.Object sourceArray)
```

Signature: VB.NET

```
Public Function CreateDataValidation(ByVal allowedType As ValidationType, ByVal sourceArray As Object)
As DataValidation
```

Parameters

allowedType

The data type allowed in the cell.

sourceArray

For 'List' allowed type: the array of valid values for the cell.

Returns

A DataValidation object.

Examples

C#

```
DataValidation dataValidation =
                wb.createDataValidation(
                DataValidation.ValidationType.List,
                listArray);
```

vb.net

```
Dim dataValidation As DataValidation = _
                wb.createDataValidation( _
                DataValidation.ValidationType.List, _
                listArray)
```

Workbook.CreateDataValidation(DataValidation.ValidationType, String)

Description

Creates a DataValidation object which can be set on a Range, Area, or Cell.

Signature: C#

```
public DataValidation CreateDataValidation(ValidationType allowedType, System.String formula)
```

Signature: VB.NET

```
Public Function CreateDataValidation(ByVal allowedType As ValidationType, ByVal formula As String) As
DataValidation
```

Parameters

allowedType

The data type allowed in the cell.

formula

For 'Custom' allowed type: the formula used to validate the cell's data. For 'List' allowed type: the formula referencing a range in the worksheet.

Returns

A DataValidation object.

Examples

C#

```
DataValidation dataValidation =
                wb.createDataValidation(
                DataValidation.ValidationType.List,
                "C10:C18");
```

vb.net

```
Dim dataValidation As DataValidation = _
               wb.createDataValidation( _
               DataValidation.ValidationType.List, _
               "C10:C18")
```

Workbook.CreateNamedRange

Overload	Description
CreateNamedRange(Area(), String)	Creates a Range of cells from an array of Areas. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.
CreateNamedRange(String, String)	Creates a named Range of cells from a specified formula. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.

Workbook.CreateNamedRange(Area(), String)

Description

Creates a Range of cells from an array of Areas. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.

Signature: C#

```
public Range CreateNamedRange(Area[] rangeAreas, System.String rangeName)
```

Signature: VB.NET

```
Public Function CreateNamedRange(ByVal rangeAreas As Area(), ByVal rangeName As String) As Range
```

Parameters

rangeAreas

Array or Areas representing the range.

rangeName

The name to assign to the range.

Returns

A `Range` object representing the range of cells created.

Examples

C#

```
Range rng = wb.CreateNamedRange(
               new Area[]{
               worksheet1.CreateArea(2, 4, 2, 5),
               worksheet2.CreateArea(2, 4, 2, 5)},
               "MyRange");
```

vb.net

```
Dim rng As Range = wb.CreateNamedRange( _
               New Area(){ _
               worksheet1.CreateArea(2, 4, 2, 5), _
               worksheet2.CreateArea(2, 4, 2, 5)}, _
               "MyRange")
```

Workbook.CreateNamedRange(String, String)

Description

Creates a named Range of cells from a specified formula. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.

Signature: C#

```
public Range CreateNamedRange(System.String rangeFormula, System.String rangeName)
```

Signature: VB.NET

```
Public Function CreateNamedRange(ByVal rangeFormula As String, ByVal rangeName As String) As Range
```

Parameters

rangeFormula

Formula representing the range, for example "=Sheet1!A1:G10". The formula must be three-dimensional (specify a sheet), and can include only sheet and cell references, and operators joining those references (range ':', intersection ' ', and union ','). Functions that return ranges cannot be used.

rangeName

The name to assign to the range.

Returns

Range A `Range` object representing the range of cells created.

Examples

C#

```
Range rng = wb.CreateNamedRange("=Sheet1!A1:G10", "Range1");
```

vb.net

```
Dim rng As Range = wb.CreateNamedRange("=Sheet1!A1:G10", "Range1")
```

Workbook.CreateNamedStyle(String)

Description

Creates a named global style, which can be bound to cells, areas, ranges, rows, and columns. Any later changes to this style will affect all cells to which the style was assigned.

Signature: C#

```
public NamedStyle CreateNamedStyle(System.String styleName)
```

Signature: VB.NET

```
Public Function CreateNamedStyle(ByVal styleName As String) As NamedStyle
```

Parameters

styleName

The name of the new style.

Returns

A NamedStyle object which can be assigned to cells, and can be accessed and modified from the generated Excel spreadsheet.

Remarks

To bind a style to a cell or set of cells, use either `Style` or `ApplyStyle`. Both are accessible through the following objects: Cell, Area, Range, RowProperties, and ColumnProperties.

Style definitions created by CreateNamedStyle are saved in the generated spreadsheet, and can be accessed and modified when the file is opened in Microsoft Excel. If a style with the same name already exists an exception will be thrown.

Examples

C#

```
NamedStyle heading1Style = wb.CreateNamedStyle("Heading1");
```

vb.net

```
Dim heading1Style As NamedStyle = wb.CreateNamedStyle("Heading1")
```

Workbook.CreateRange

Overload	Description
CreateRange(Area())	Creates a Range of cells from an array of Areas. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.
CreateRange(String)	Creates a Range of cells from a specified formula. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.

Workbook.CreateRange(Area())

Description

Creates a Range of cells from an array of Areas. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.

Signature: C#

```
public Range CreateRange(Area[] rangeAreas)
```

Signature: VB.NET

```
Public Function CreateRange(ByVal rangeAreas As Area()) As Range
```

Parameters

rangeAreas

Array or Areas representing the range.

Returns

A `Range` object representing the range of cells created.

Examples

C#

```
Range rng = wb.CreateRange(
               new Area[]{
               worksheet1.CreateArea(2, 4, 2, 5),
               worksheet2.CreateArea(2, 4, 2, 5)});
```

vb.net

```
Dim rng As Range = wb.CreateRange( _
               New Area(){ _
               worksheet1.CreateArea(2, 4, 2, 5), _
               worksheet2.CreateArea(2, 4, 2, 5)})
```

Workbook.CreateRange(String)

Description

Creates a Range of cells from a specified formula. A `Range` is a collection of Area s; an `Area` area is a rectangular collection of cells.

Signature: C#

```
public Range CreateRange(System.String rangeFormula)
```

Signature: VB.NET

```
Public Function CreateRange(ByVal rangeFormula As String) As Range
```

Parameters

rangeFormula

Formula representing the range, for example "=Sheet1!A1:G10". The formula must be three-dimensional (specify a sheet), and can include only sheet and cell references, and operators joining those references (range ':', intersection ' ', and union ','). Functions that return ranges cannot be used.

Returns

A `Range` object representing the range of cells created.

Examples

C#

```
Range rng = wb.CreateRange("=Sheet1!A1:G10");
```

vb.net

```
Dim rng As Range = wb.CreateRange("=Sheet1!A1:G10")
```

Workbook.CreateStyle()

Description

Creates a global style, which can be bound to cells, areas, ranges, rows, and columns. Any later changes to this style will affect all cells to which the style was bound.

Signature: C#

```
public GlobalStyle CreateStyle()
```

Signature: VB.NET

```
Public Function CreateStyle() As GlobalStyle
```

Returns

A GlobalStyle object which can be linked and applied to cells, but will not be accessible from the generated Excel spreadsheet.

Remarks

To bind a style to a cell or set of cells, use either Style or `ApplyStyle`. Both are accessible through the following objects: Cell, Area, Range, RowProperties, and ColumnProperties.

Examples

C#

```
Style styl = wb.CreateStyle();
```

vb.net

```
Dim styl As Style = wb.CreateStyle()
```

Workbook.DisplayTabs

Description

Shows or hides the worksheet tabs at the bottom of the workbook window.

Signature: C#

```
public boolean DisplayTabs{ get; set; }
```

Signature: VB.NET

```
Public Property DisplayTabs() As Boolean
```

Examples

C#

```
//--- Get the DisplayTabs value of the workbook
          bool tabs = wb.DisplayTabs;

          //--- Set the DisplayTabs value of the workbook
          wb.DisplayTabs = true;
```

vb.net

```
'--- Get the DisplayTabs value of the workbook
          Dim tabs As Boolean = wb.DisplayTabs

          '--- Set the DisplayTabs value of the workbook
          wb.DisplayTabs = True
```

Workbook.DocumentProperties

Description

Returns a DocumentProperties object that represents properties of the specified workbook, such as name, author, and summary. In Microsoft Excel, these properties are displayed in the file's Properties sheet. To view the Properties sheet in Excel, open Excel's File menu, and select Properties.

Signature: C#

```
public DocumentProperties DocumentProperties{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property DocumentProperties() As DocumentProperties
```

Examples

C#

```
DocumentProperties docProps =
               wb.DocumentProperties;
```

vb.net

```
Dim docProps As DocumentProperties = _
               wb.DocumentProperties
```

Workbook.EncryptPassword

Description

Sets or returns the password used to encrypt the spreadsheet. Standard "Office 97/2000 Compatible" encryption is used to encrypt the spreadsheet. This is the default encryption used by Excel 97 and later versions.

Signature: C#

```
public System.String EncryptPassword{ get; set; }
```

Signature: VB.NET

```
Public Property EncryptPassword() As String
```

Remarks

To further protect your spreadsheet, you can protect the workbook structure using the Workbook.Protect() method or write-protect individual worksheets using the Worksheet.Protect() method.

Examples

C#

```
//--- Get the EncryptPassword value for the workbook
          string pwd = wb.EncryptPassword;

          //--- Set the EncryptPassword value for the workbook
          wb.EncryptPassword = "MyPassword";
```

vb.net

```
'--- Get the EncryptPassword value for the workbook
          Dim pwd As String = wb.EncryptPassword

          '--- Set the EncryptPassword value for the workbook
          wb.EncryptPassword = "MyPassword"
```

Workbook.ExcludeMacros

Description

Sets or returns whether ExcelWriter should remove macros from the generated workbook or leave them in.

Signature: C#

```
public boolean ExcludeMacros{ get; set; }
```

Signature: VB.NET

```
Public Property ExcludeMacros() As Boolean
```

Remarks

When `ExcludeMacro` is set to true, buttons associated with macros may produce the error "Data may have been lost." Therefore, when excluding macros, remove all associated buttons from the spreadsheet. ExcelWriter, by default, will include macros in the generated workbook.

Examples

C#

```
//--- Get the ExcludeMacros value for the workbook
          bool exclude = wb.ExcludeMacros;

          //--- Set the ExcludeMacros value for the workbook
          wb.ExcludeMacros = true;
```

vb.net

```
'--- Get the ExcludeMacros value for the workbook
          Dim exclude As Boolean = wb.ExcludeMacros

          '--- Set the ExcludeMacros value for the workbook
          wb.ExcludeMacros = True
```

Workbook.FirstDisplayedTab

Description

Sets or returns the first worksheet tab to display in the tab bar at the bottom of the workbook window.

Signature: C#

```
public int FirstDisplayedTab{ get; set; }
```

Signature: VB.NET

```
Public Property FirstDisplayedTab() As Integer
```

Examples

C#

```
//--- Get the FirstDisplayedTab value of the workbook
          int firstTab = wb.FirstDisplayedTab;

          //--- Set the FirstDisplayedTab value of the workbook
          wb.FirstDisplayedTab = 0;
```

vb.net

```
'--- Get the FirstDisplayedTab value of the workbook
          Dim firstTab As Integer = wb.FirstDisplayedTab

          '--- Set the FirstDisplayedTab value of the workbook
          wb.FirstDisplayedTab = 0
```

Workbook.GetEnumerator()

Description

Returns an IEnumerator for the Worksheets in a Workbook

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the Worksheets in a Workbook as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = wb.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = wb.GetEnumerator()
```

Workbook.GetNamedObject(String)

Description

Returns the NamedObject object that represents the name of a specified array, number, picture, or range. If the named object does not exist, the method returns null.

Signature: C#

```
public NamedObject GetNamedObject(System.String name)
```

Signature: VB.NET

```
Public Function GetNamedObject(ByVal name As String) As NamedObject
```

Parameters

name

The string representation of the object's name.

Returns

The `NamedObject` object of a specified array, number, picture, or range, or null if the named object does not exist.

Examples

C#

```
NamedObject obj = wb.GetNamedObject("MyObject");
```

vb.net

```
Dim obj As NamedObject = wb.GetNamedObject("MyObject")
```

Workbook.GetNamedRange(String)

Description

Returns the specified named Range. If the named range does not exist, the method returns null.

Signature: C#

```
public Range GetNamedRange(System.String name)
```

Signature: VB.NET

```
Public Function GetNamedRange(ByVal name As String) As Range
```

Parameters

name

The name of the range.

Returns

A named `Range` object, or null if the range does not exist.

Examples

C#

```
Range rng = wb.GetNamedRange("Range1");
```

vb.net

```
Dim rng As Range = wb.GetNamedRange("Range1")
```

Workbook.GetNamedStyle(String)

Description

Returns a NamedStyle if it exists. If the style does not exist, null is returned.

Signature: C#

```
public NamedStyle GetNamedStyle(System.String styleName)
```

Signature: VB.NET

```
Public Function GetNamedStyle(ByVal styleName As String) As NamedStyle
```

Parameters

styleName

The name of the style to retrieve.

Returns

The named style with the name `styleName`, or null if the style exists.

Examples

C#

```
NamedStyle heading1Style = wb.GetNamedStyle("Heading1");
```

vb.net

```
Dim heading1Style As NamedStyle = wb.GetNamedStyle("Heading1")
```

Workbook.GetTextHeight(String, Font, Double)

Description

Returns the minimum height required to fit the given text string using the given font.

Signature: C#

```
public double GetTextHeight(System.String text, Font font, double desiredWidth)
```

Signature: VB.NET

```
Public Function GetTextHeight(ByVal text As String, ByVal font As Font, ByVal desiredWidth As Double)
As Double
```

Parameters

text

The text string whose height will be calculated.

font

The font that will be used to format the text.

desiredWidth

The width, in character units, of the cell or area into which the text will be inserted.

Returns

The minimum height, in character units, required to accommodate the given text string.

Remarks

The maximum height of any one row is 409 units, so if the height return is greater than 409, you must merge cells across multiple rows and set the height of each row to accommodate the total minimum height of the text.

Examples

C#

```
//--- What is the width of column C?
          ColumnProperties colProps = worksheet.GetColumnProperties(2);

          //--- How tall will the data be?
          double requiredHeight =
               wb.GetTextHeight(longTextString,
               colProps.Style.Font,
               colProps.WidthInChars);

          //--- Will we need multiple rows to display all of the data?
          //--- Can not exceed 409 character units in height per row.
          int numRows = (int) Math.Round((requiredHeight / 409) + 0.5);

          //--- Evenly distribute the required height across all of the rows.
          worksheet.Cells[2, 2].Value = longTextString;
          Area a = worksheet.CreateArea(2, 2, numRows, 1);
          a.MergeCells();
          a.AllRowsHeight = (requiredHeight / numRows);
```

vb.net

```
'--- What is the width of column C?
          Dim colProps As ColumnProperties = worksheet.GetColumnProperties(2)

          '--- How tall will the data be?
          Dim requiredHeight As Double = _
               wb.GetTextHeight(longTextString, _
               colProps.Style.Font, _
               colProps.WidthInChars)

          '--- Will we need multiple rows to display all of the data?
          '--- Can not exceed 409 character units in height per row.
          Dim numRows As Integer = Math.Round((requiredHeight / 409) + 0.5)

          '--- Evenly distribute the required height across all of the rows.
          worksheet.Cells(2, 2).Value = longTextString
          Dim a As Area = worksheet.CreateArea(2, 2, numRows, 1)
          a.MergeCells()
          a.AllRowsHeight = (requiredHeight / numRows)
```

Workbook.GetTextWidth(String, Font, Double)

Description

Returns the minimum width required to fit the given text string using the given font.

Signature: C#

```
public double GetTextWidth(System.String text, Font font, double desiredHeight)
```

Signature: VB.NET

```
Public Function GetTextWidth(ByVal text As String, ByVal font As Font, ByVal desiredHeight As Double)
As Double
```

Parameters

text

The text string whose width will be calculated.

font

The font that will be used to format the text.

desiredHeight

The height, in character units, of the cell or area into which the text will be inserted.

Returns

The minimum width, in character units, required to accomodate the given text string.

Remarks

The maximum width of any one column is 409 character units, so if the width return is greater than 409, you must merge cells across multiple columns and set the width of each column to accomodate the total minimum width of the text.

Examples

C#

```
//--- What is the height of row 1?
          RowProperties rowProps = worksheet.GetRowProperties(0);

          //--- How wide will the data be?
          double requiredWidth =
               wb.GetTextWidth(longTextString,
               rowProps.Style.Font,
               rowProps.Height);

          //--- Will we need multiple columns to display all of the data?
          //--- Can not exceed 255 character units in width per column.
          int numCols = (int) Math.round((requiredWidth / 255) + 0.5);

          //--- Evenly distribute the required width between all of the columns.
          worksheet.Cells[0, 3].Value = longTextString;
          a = worksheet.createArea(0, 3, 1, numCols);
          a.MergeCells();
          a.AllColumnWidthsInChars = (requiredWidth / numCols);
```

vb.net

```
'--- What is the height of row 1?
          Dim rowProps As RowProperties = worksheet.GetRowProperties(0)

          '--- How wide will the data be?
          Dim requiredWidth As Double = _
               wb.GetTextWidth(longTextString, _
               rowProps.Style.Font, _
               rowProps.Height)

          '--- Will we need multiple columns to display all of the data?
          '--- Can not exceed 409 character units in width per column.
          Dim numCols As Integer = Math.Round((requiredWidth / 409) + 0.5)

          '--- Evenly distribute the required width between all of the columns.
          worksheet.Cells(0, 3).Value = longTextString
          Dim a As Area = worksheet.CreateArea(2, 2, numRows, 1)
          a.MergeCells()
          a.AllColumnWidthsInChars = (requiredHeight / numRows)
```

Workbook.Height

Description

Sets or returns the height of the workbook's window

Signature: C#

```
public int Height{ get; set; }
```

Signature: VB.NET

```
Public Property Height() As Integer
```

Examples

C#

```
//--- Get the height of the workbook window
          int winHeight = wb.Height;

          //--- Set the height of the workbook window
          wb.Height = 500;
```

vb.net

```
'--- Get the height of the workbook window
          Dim winHeight As Integer = wb.Height

          '--- Set the height of the workbook window
          wb.Height = 500
```

Workbook.HorizontalPosition

Description

Sets or returns the horizontal (x) position of the workbook window's upper left corner when displayed in Excel.

Signature: C#

```
public int HorizontalPosition{ get; set; }
```

Signature: VB.NET

```
Public Property HorizontalPosition() As Integer
```

Examples

C#

```
//--- Get the horizontal position of the workbook window's upper left corner
          int winHoriz = wb.HorizontalPosition;

          //--- Set the horizontal position of the workbook window's upper left corner
          wb.HorizontalPosition = 100;
```

vb.net

```
'--- Get the horizontal position of the workbook window's upper left corner
          Dim winHoriz As Integer = wb.HorizontalPosition

          '--- Set the horizontal position of the workbook window's upper left corner
          wb.HorizontalPosition = 100
```

Workbook.IsProtected

Description

Returns whether or not the worksheet is protected.

Signature: C#

```
public boolean IsProtected{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property IsProtected() As Boolean
```

Examples

C#

```
bool protected = wb.IsProtected;
```

vb.net

```
Dim protected As Boolean = wb.IsProtected
```

Workbook.Item

Overload	Description
Item(Int32)	Returns the worksheet specified by a 0-based index. This property is an indexer for the Workbook class.
Item(String)	Returns the worksheet with the specified name. This property is an indexer for the Workbook class.

Workbook.Item(Int32)

Description

Returns the worksheet specified by a 0-based index. This property is an indexer for the Workbook class.

Signature: C#

```
public Worksheet this[int sheetOffset] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal sheetOffset As Integer) As Worksheet
```

Parameters

sheetOffset

The 0-based index of the worksheet. The first worksheet is `Worksheet[0]`.

Returns

The worksheet at the specified index.

Examples

C#

```
//--- Get Worksheet at position 0
          Worksheet ws = wb[0];
```

vb.net

```
'--- Get Worksheet at position 0
          Dim ws As Worksheet = wb(0)
```

Workbook.Item(String)

Description

Returns the worksheet with the specified name. This property is an indexer for the Workbook class.

Signature: C#

```
public Worksheet this[System.String sheetName] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal sheetName As String) As Worksheet
```

Parameters

name

The name of the worksheet.

Returns

The worksheet with the specified name.

Examples

C#

```
//--- Get the worksheet titled "Sheet1"
          Worksheet ws = wb["Sheet1"];
```

vb.net

```
'--- Get the worksheet titled "Sheet1"
          Dim ws As Worksheet = wb("Sheet1")
```

Workbook.NamedRanges

Description

Returns the named ranges of the Workbook as an array of NamedObject objects as an `IEnumerable` collection. Though the named ranges are not returned in any particular order, they iterable.

Signature: C#

```
public System.Collections.Generic.IEnumerable<NamedObject> NamedRanges{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NamedRanges() As System.Collections.Generic.IEnumerable(Of NamedObject)
```

Examples

C#

```
NamedObject[] ranges = wb.NamedRanges;
```

vb.net

```
Dim ranges as NamedObject() = wb.NamedRanges
```

Workbook.NumberFormat

Description

Returns the NumberFormat used for formatting of numbers and dates.

Signature: C#

```
public NumberFormat NumberFormat{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NumberFormat() As NumberFormat
```

Examples

C#

```
NumberFormat format = wb.NumberFormat;
```

vb.net

```
Dim format As NumberFormat = wb.NumberFormat
```

Workbook.Palette

Description

Returns a Palette object that represents the specified workbook's color palette.

Signature: C#

```
public Palette Palette{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Palette() As Palette
```

Remarks

Each Excel workbook has a single color palette that manages formatting colors and some drawing layer colors. A color palette contains 56 colors that may be referenced. Each of these 56 colors may be customized, but changing a color in the palette will affect all formatting that uses the referenced color.

Examples

C#

```
Palette pal = wb.Palette;
```

vb.net

```
Dim pal As Palette = wb.Palette
```

Workbook.PasswordToModify

Description

This sets the workbook so that it can be opened in read-only mode with the option to open in read-write mode, if the user supplies a matching password when opening the spreadsheet.

Signature: C#

```
public System.String PasswordToModify{ set; }
```

Signature: VB.NET

```
Public WriteOnly Property PasswordToModify() As String
```

Examples

C#

```
wb.PasswordToModify = "MyPassword";
```

vb.net

```
wb.PasswordToModify = "MyPassword"
```

Workbook.Protect(String)

Description

Protects the workbook structure using the supplied password. End users will not be able to add or remove worksheets unless they enter the password.

Signature: C#

```
public void Protect(System.String password)
```

Signature: VB.NET

```
Public Sub Protect(ByVal password As String)
```

Parameters

password

The password to use to protect the workbook. If the password = "" or null, then the workbook will be protected without a password.

Remarks

The equivalent to this action in Excel is to select "Protect structure and windows" from the review menu.

Further protection can be extended to cells by using the Worksheet.Protect() method to protect individual worksheets and by using the Workbook.EncryptPassword property to set a password that must be provided to open the workbook.

Examples

C#

```
wb.Protect("MyPassword");
```

vb.net

```
wb.Protect("MyPassword")
```

Workbook.ProtectPasswordHash

Description

Sets or returns the password hash that is used to protect the workbook.

Signature: C#

```
public int ProtectPasswordHash{ get; set; }
```

Signature: VB.NET

```
Public Property ProtectPasswordHash() As Integer
```

Examples

C#

```
//--- Get the ProtectPasswordHash value for the workbook
          int passwordHash = wb.ProtectPasswordHash;

          //--- Set the ProtectPasswordHash value for the workbook
          wb.ProtectPasswordHash = 12345
```

vb.net

```
'--- Get the ProtectPasswordHash value for the workbook
          Dim passwordHash As Integer = wb.ProtectPasswordHash

          '--- Set the ProtectPasswordHash value for the workbook
          wb.ProtectPasswordHash = 12345
```

Workbook.ReadOnly

Description

Determines whether the workbook can only be opened in read-only mode.

Signature: C#

```
public boolean ReadOnly{ get; set; }
```

Signature: VB.NET

```
Public Property [ReadOnly]() As Boolean
```

Examples

C#

```
//--- Get the ReadOnly value for the workbook
          bool readOnly = wb.ReadOnly;

          //--- Set the ReadOnly value for the workbook
          wb.ReadOnly = true;
```

vb.net

```
'--- Get the ReadOnly value for the workbook
          Dim readOnly As Boolean = wb.ReadOnly

          '--- Set the ReadOnly value for the workbook
          wb.ReadOnly = True
```

Workbook.ReadOnlyRecommended

Description

Sets or returns whether Excel will recommend - when the workbook is opened - that it be viewed in read-only mode.

Signature: C#

```
public boolean ReadOnlyRecommended{ get; set; }
```

Signature: VB.NET

```
Public Property ReadOnlyRecommended() As Boolean
```

Examples

C#

```
//--- Get the ReadOnlyRecommended value for the workbook
          bool readOnlyRec = wb.ReadOnlyRecommended;

          //--- Set the ReadOnlyRecommended value for the workbook
          wb.ReadOnlyRecommended = true;
```

vb.net

```
'--- Get the ReadOnlyRecommended value for the workbook
          Dim readOnlyRec As Boolean = wb.ReadOnlyRecommended

          '--- Set the ReadOnlyRecommended value for the workbook
          wb.ReadOnlyRecommended = True
```

Workbook.ShowHorizontalScrollBar

Description

When true, the horizontal scroll bar will be shown in the generated workbook. When false, the horizontal scroll bar will be hidden.

Signature: C#

```
public boolean ShowHorizontalScrollBar{ get; set; }
```

Signature: VB.NET

```
Public Property ShowHorizontalScrollBar() As Boolean
```

Examples

C#

```
//--- Set the ShowHorizontalScrollBar property value
          bool showHorizScroll = wb.ShowHorizontalScrollBar;

          //--- Get the ShowHorizontalScrollBar property value
          wb.ShowHorizontalScrollBar = true;
```

vb.net

```
'--- Set the ShowHorizontalScrollBar property value
          Dim showHorizScroll As Boolean = wb.ShowHorizontalScrollBar

          '--- Get the ShowHorizontalScrollBar property value
          wb.ShowHorizontalScrollBar = True
```

Workbook.ShowVerticalScrollBar

Description

Shows or hides the vertical scroll bar in the generated workbook.

Signature: C#

```
public boolean ShowVerticalScrollBar{ get; set; }
```

Signature: VB.NET

```
Public Property ShowVerticalScrollBar() As Boolean
```

Examples

C#

```
//--- Get the ShowVerticalScrollBar property
          bool showVertScroll = wb.ShowVerticalScrollBar;

          //--- Set the ShowVerticalScrollBar property
          wb.ShowVerticalScrollBar = true;
```

vb.net

```
'--- Get the ShowVerticalScrollBar property
          Dim showVertScroll As Boolean = wb.ShowVerticalScrollBar

          '--- Set the ShowVerticalScrollBar property
          wb.ShowVertialScrollBar = True
```

Workbook.TabRatio

Description

Sets or returns the ratio between the workbook's tab bar and the horizontal scroll bar.

Signature: C#

```
public double TabRatio{ get; set; }
```

Signature: VB.NET

```
Public Property TabRatio() As Double
```

Remarks

For example, if TabRatio is set to 50, the tab bar will take up 50% of the width of the workbook window, while the horizontal scroll bar will take up the other half.The value of `TabRatio` should be between 0 and 100. If the value is less than 0, ExcelWriter will set `TabRatio` to 0. If the value is greater than 100, ExcelWriter will set `TabRatio` to 100.

Examples

C#

```
//--- Get the ratio between the workbook's tab bar and the horizontal scroll bar
          double ratio = wb.TabRatio;

          //--- Set the ratio between the workbook's tab bar and the horizontal scroll bar
          wb.TabRatio = 20;
```

vb.net

```
'--- Get the ratio between the workbook's tab bar and the horizontal scroll bar
          Dim ratio As Double = wb.TabRatio

          '--- Set the ratio between the workbook's tab bar and the horizontal scroll bar
          wb.TabRatio = 20
```

Workbook.Unprotect()

Description

Removes protection from the workbook.

Signature: C#

```
public void Unprotect()
```

Signature: VB.NET

```
Public Sub Unprotect()
```

Examples

C#

```
wb.Unprotect();
```

vb.net

```
wb.Unprotect()
```

Workbook.UseRCFormulaNotation

Description

Indicates or sets the type of references used in this workbook.

Signature: C#

```
public boolean UseRCFormulaNotation{ get; set; }
```

Signature: VB.NET

```
Public Property UseRCFormulaNotation() As Boolean
```

Remarks

When false, the default, all references will be in "Normal" style, and should be expressed as A1, $A$1, etc. If set to true, references will be in RC style, and should be expressed as R1C1, R[1]C[1], etc. All methods and properties that take formulas as input or output will pay attention to this value. Setting this value will also affect the style of references that Excel displays on opening the file. This option is analagous to the Tools->Options->General->R1C1 Reference Style in Excel.

Examples

C#

```
wb.UseRCFormulaNotation = true;
          myCell.Formula = "=R3C2";
```

vb.net

```
wb.UseRCFormulaNotation = True
          myCell.Formula = "=R3C2"
```

Workbook.VerticalPosition

Description

Sets or returns the vertical (y) position of the workbook window's upper left corner when displayed in Excel.

Signature: C#

```
public int VerticalPosition{ get; set; }
```

Signature: VB.NET

```
Public Property VerticalPosition() As Integer
```

Examples

C#

```
//--- Get the vertical position of the workbook window's upper left corner
          int winVert = wb.VerticalPosition;

          //--- Set the vertical position of the workbook window's upper left corner
          wb.VerticalPosition = 100;
```

vb.net

```
'--- Get the vertical position of the workbook window's upper left corner
          Dim winVert As Integer = wb.VerticalPosition

          '--- Set the vertical position of the workbook window's upper left corner
          wb.VerticalPosition = 100
```

Workbook.Width

Description

Sets or returns the width of the workbook's window.

Signature: C#

```
public int Width{ get; set; }
```

Signature: VB.NET

```
Public Property Width() As Integer
```

Examples

C#

```
//--- Get the width of the workbook window
          int winWidth = wb.Width;

          //--- Set the width of the workbook window
          wb.Width = 500;
```

vb.net

```
'--- Get the width of the workbook window
          Dim winWidth As Integer = wb.Width

          '--- Set the width of the workbook window
          wb.Width = 500
```

Workbook.Worksheets

Description

Returns a Worksheets collection which provides access to all worksheets in the specified workbook, including charts displayed as sheets.

Signature: C#

```
public Worksheets Worksheets{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Worksheets() As Worksheets
```

Remarks

Use the `Worksheets` object to manage the worksheets in a workbook.
ExcelWriter may not be able to parse some worksheets (such as macro sheets).

Examples

C#

```
Worksheets wkshts = wb.Worksheets;
```

vb.net

```
Dim wkshts As Worksheets = wb.Worksheets
```

Worksheet

Description

The Worksheet class represents a single Excel worksheet.

Signature: C#

```
[DefaultMember("Item")]
 public class Worksheet
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public Class Worksheet
```

Remarks

To add a Worksheet to a Workbook, call Workbook.Worksheets.CreateWorksheet(). To get an existing Worksheet call Workbook.Worksheets[index or name].

Examples

C#

```
//--- Create a new Worksheet
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets.CreateWorksheet("Sheet2");

          //--- Get an existing Worksheet by index
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheets ws = wb.Worksheets[0];

          //--- Get an existing Worksheet by name
          ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          Worksheet ws = wb.Worksheets["Sheet1"];
```

vb.net

```
'--- Create a new Worksheet
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets.CreateWorksheet("Sheet2")

          '--- Get an existing Worksheet by index
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets(0)

          '--- Get an existing Worksheet by name
          Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim ws As Worksheet = wb.Worksheets("Sheet2")
```

Properties

Name	Description
Cells	Returns a Cells collection that contains all cells in the worksheet.
Charts	Returns a Charts collection containing all the charts contained as objects within this worksheet. Use this collection to add, access, and remove charts from the worksheet.
Comments	Returns a Comments collection that contains all comments in the worksheet.
FirstShownColumn	Sets or returns the 0-based index of the first column that is shown in the worksheet.
FirstShownRow	Sets or returns the 0-based index of the first row that is shown in the worksheet.
FreezePanes	Freezes panes in a worksheet or returns the cell at which the panes are split. Set the property to a Cell object. The worksheet will be split above the cell, to the left of it, or both.
GridlinesColor	Sets or returns a `Color` object representing the color of the gridlines in the worksheet.
Hyperlinks	Returns the collection of hyperlinks in the worksheet.
IsProtected	Returns whether or not the worksheet is write-protected.
IsSelected	Returns whether this worksheet is selected. Multiple worksheets can be selected.
Name	Sets or returns the name of the worksheet. Sheet names are limited to 255 characters and must not contain the following characters: \ / ? [ ] : '
NamedRanges	Returns the named ranges of the Worksheet as an array of NamedObject objects as an `IEnumerable` collection. Though the named ranges are not returned in any particular order, they iterable.
PageSetup	Returns a PageSetup object representing page layout properties of a printed worksheet.
Pictures	Returns a Pictures collection containing all pictures in the worksheet. Use this collection to add, access, and remove pictures from the worksheet.
PopulatedCells	Returns a rectanglular area of cells from the first (top, left) populated cell to the last (bottom, right) populated cell in the worksheet. The area returned will include cells that contain no data but have been formatted.
Position	Returns the 0-based index of the worksheet in the collection of all sheets in the workbook.
ProtectPasswordHash	Sets or returns the password hash that is used to write-protect (not encrypt) the worksheet.
ShapeGroups	Returns a Groups collection containing all shape groups within the worksheet.
Shapes	Returns a Shapes collection containing all shapes within the worksheet.
ShowFormulas	Sets or returns whether formulas or their calculated values will be displayed in the worksheet. Calculated values are displayed by default.
ShowGridlines	Sets or returns whether gridlines should be shown in the worksheet. Gridlines are shown by default.
ShowRowColHeaders	Sets or returns whether column and row headers will be displayed in the worksheet. The headers are displayed by default.

ShowZeroValues	Sets or returns whether zero values in cells will be displayed in the worksheet. Zero values are displayed by default.
StandardHeight	Sets or returns the height of the "Standard Row" in points (1/72 of an inch). This must be a value that when converted to character widths is between 0 - 409.
StandardWidth	Sets or returns the width of the "Standard Column", in points (1/72 of an inch). This must be a value that when converted to character widths is between 0 - 255.
StandardWidthInChars	Sets or returns the width of the "Standard Column" as a number of '0' characters in the font of the 'Normal' style. This must be a value between 0 - 255.
SummaryColumns	When grouping columns, SummaryColumns determines where the summary columns for the groups are located. By default the summary rows are located to the right of the groupings.
SummaryRows	When grouping rows, SummaryRows determines where the summary rows for the groups are located. By default the summary rows are located below the groupings.
TabColor	Sets or returns the color of the worksheet's sheet name tab.
Workbook	Returns the Workbook object in which this worksheet resides.
ViewState	Sets or returns the worksheet's viewing mode, which may be either normal or page break mode.
Visibility	Sets or returns the visibility setting of the worksheet.
ZoomPercentage	Sets or returns the percentage of standard size by which the worksheet will be magnified or reduced. Set this property to a value between 10 and 400.

Indexers

Name	Description
Item(Int32, Int32)	Returns the cell at a specified row and column position. This property is an indexer for the Worksheet class.
Item(String)	Returns the cell at the specified Excel-style reference. This property is an indexer for the Worksheet class.

Methods

Name	Description
CopyPaste(Cell, Area)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(Cell, Area, CopyPasteProperties)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(Int32, Int32, Area)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(Int32, Int32, Area, CopyPasteProperties)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(String, Area, CopyPasteProperties)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(String, Area)	Copies an area of cells from another Worksheet to cells in this worksheet.
CreateAnchor(Int32, Int32, Double, Double)	Creates an anchor at a specified position in the worksheet.
CreateArea(Int32, Int32, Int32, Int32)	Defines a rectangular Area of cells in the worksheet.
CreateArea(String)	Defines a rectangular Area of cells in the worksheet.
CreateAreaOfColumns(Int32, Int32)	Returns an Area object representing all the cells in a specified group of columns. An area is a rectangular collection of cells.
CreateAreaOfRows(Int32, Int32)	Returns an Area object representing all the cells in a specified group of rows. An area is a rectangular collection of cells.
CreateNamedRange(Int32, Int32, Int32, Int32, String)	Creates a named range, containing one area. A range is a collection of areas; an area is a rectangular collection of cells.
CreateNamedRange(String, String)	Creates a named range from a specified formula. A range is a collection of areas; an area is a rectangular collection of cells. This method can be used to create a non-rectangular range containing multiple rectangular areas.

CreateRange(String)	Creates a range from a specified formula. A range is a collection of areas; an area is a rectangular collection of cells. This method can be used to create a non-rectangular range containing multiple rectangular areas.
DeleteColumn(Int32)	Deletes a column and its contents from the worksheet. Columns after the deleted column will be moved to the left.
DeleteColumns(Int32, Int32)	Deletes a number of columns and its contents from the worksheet. Columns after the deleted columns will be moved to the left.
DeleteRow(Int32)	Deletes a specified row and its contents from the worksheet. Rows below the deleted row will be moved up.
DeleteRows(Int32, Int32)	Deletes a specified number of rows and its contents from the worksheet. Rows below the deleted rows will be moved up.
GetColumnProperties(Int32)	Returns a ColumnProperties object representing the column specified by index.
GetNamedObject(String)	Returns the NamedObject object that represents the name of a specified array, number, picture, or range. If the named object does not exist, the method returns null.
GetNamedRange(String)	Returns the named Range object that specified by name. If the named range does not exist, the method returns null.
GetRowProperties(Int32)	Returns a RowProperties object representing the row specified by index.
GroupColumns(Int32, Int32, Boolean)	Groups or outlines a contiguous set of columns.
GroupRows(Int32, Int32, Boolean)	Groups or outlines a contiguous set of rows.
ImportData(Object()(), Cell)	Imports data from a two-dimensional array of objects to cells in the worksheet.
ImportData(Object()(), String(), Cell, DataImportProperties)	Imports data from a two-dimensional array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataTable, Cell)	Imports data from an ADO.NET `DataTable` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataTable, Cell, DataImportProperties)	Imports data from an ADO.NET `DataTable` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataView, Cell)	Imports data from an ADO.NET `DataView` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataView, Cell, DataImportProperties)	Imports data from an ADO.NET `DataView` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object(,), Cell)	Imports data from a rectangular array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object(,), String(), Cell, DataImportProperties)	Imports data from a rectangular array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.IDataReader, Cell)	Imports data from an `IDataReader` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.IDataReader, Cell, DataImportProperties)	Imports data from an `IDataReader` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
InsertColumn(Int32)	Inserts a column in the worksheet to the left of the specified column.
InsertColumn(Int32, ColumnInsertBehavior)	Inserts a column in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.
InsertColumn(Int32, ColumnInsertBehavior, ColumnInsertBehavior)	Inserts a column in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.
InsertColumns(Int32, Int32)	Inserts a block of columns in the worksheet to the left of the specified column.
InsertColumns(Int32, Int32, ColumnInsertBehavior)	Inserts a block of columns in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.
InsertColumns(Int32, Int32, ColumnInsertBehavior, ColumnInsertBehavior)	Inserts a block of columns in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.

InsertHorizontalPageBreak(Cell)	Inserts a horizontal page break in the worksheet after the specified cell.
InsertRow(Int32)	Inserts a row in the worksheet above the specified row.
InsertRow(Int32, RowInsertBehavior)	Inserts a row in the worksheet above the specified row.
InsertRow(Int32, RowInsertBehavior, RowInsertBehavior)	Inserts a row in the worksheet above the specified row.
InsertRows(Int32, Int32)	Inserts a number of rows in the worksheet above the specified row.
InsertRows(Int32, Int32, RowInsertBehavior)	Inserts a number of rows in the worksheet above the specified row.
InsertRows(Int32, Int32, RowInsertBehavior, RowInsertBehavior)	Inserts a number of rows in the worksheet above the specified row.
InsertVerticalPageBreak(Cell)	Inserts a vertical page break in the worksheet to right of the specified cell.
Protect(String)	Write-protects the worksheet. A user will not be able to modify the worksheet in Excel without entering the specified password. This method does not encrypt the worksheet.
Select()	Selects the current worksheet and deselects all others. To select multiple worksheets, use Worksheets.Select().
UngroupColumns(Int32, Int32)	Ungroups a contiguous set of columns.
UngroupRows(Int32, Int32)	Ungroups a contiguous set of rows.
Unprotect()	Removes the write-protection from the worksheet.

Extension Methods

Introduced in build 7.5

Overload	Description
ImportData(Cell, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint List to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Cell, Microsoft.SharePoint.SPList)	Imports data from a SharePoint List to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint View to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Imports data from a SharePoint View to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Nested Classes

Name	Description
Constants	Defines a set of constants related to worksheets.
SheetViewState	A SheetViewState value specifies the view state of a worksheet in Excel.
SheetVisibility	A SheetVisibility value specifies the visibility level of a worksheet.
SummaryColumnsLocation	A SummaryColumnsLocation value specifies the location of the summary column for a set of grouped columns.
SummaryRowsLocation	A SummaryColumnsLocation value specifies the location of the summary row for a set of grouped rows.

Worksheet.Cells

Description

Returns a Cells collection that contains all cells in the worksheet.

Signature: C#

```
public Cells Cells{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Cells() As Cells
```

Examples

C#

```
Cells cls = ws.Cells;
```

vb.net

```
Dim cls As Cells = ws.Cells
```

Worksheet.Charts

Description

Returns a Charts collection containing all the charts contained as objects within this worksheet. Use this collection to add, access, and remove charts from the worksheet.

Signature: C#

```
public Charts Charts{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Charts() As Charts
```

Examples

C#

```
Charts allCharts = ws.Charts;
```

vb.net

```
Dim allCharts As Charts = ws.Charts
```

Worksheet.Comments

Description

Returns a Comments collection that contains all comments in the worksheet.

Signature: C#

```
public Comments Comments{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Comments() As Comments
```

Examples

C#

```
Comments cmnts = ws.Comments;
```

vb.net

```
Dim cmnts As Comments = ws.Comments
```

Worksheet.Constants

Description

Defines a set of constants related to worksheets.

Signature: C#

```
public class Constants
```

Signature: VB.NET

```
Public Class Constants
```

Values

Name	Description
MAX_COLUMNS	A constant holding the maximum index of a column on a worksheet, 255.
MAX_ROWS	A constant holding the maximum index of a row on a worksheet, 65535.

Worksheet.CopyPaste

Overload	Description
CopyPaste(Cell, Area)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(Cell, Area, CopyPasteProperties)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(Int32, Int32, Area)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(Int32, Int32, Area, CopyPasteProperties)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(String, Area)	Copies an area of cells from another Worksheet to cells in this worksheet.
CopyPaste(String, Area, CopyPasteProperties)	Copies an area of cells from another Worksheet to cells in this worksheet.

Worksheet.CopyPaste(Cell, Area, CopyPasteProperties)

Description

Copies an area of cells from another Worksheet to cells in this worksheet.

Signature: C#

```
public Area CopyPaste(Cell destCell, Area sourceArea, CopyPasteProperties properties)
```

Signature: VB.NET

```
Public Function CopyPaste(ByVal destCell As Cell, ByVal sourceArea As Area, ByVal properties As
CopyPasteProperties) As Area
```

Parameters

destCell

A Cell indicating the upper-left cell of the destination area.

sourceArea

An area from the source workbook from which to copy.

properties

A CopyPasteProperties object that specifies the types of data to copy.

Returns

An Area object representing the set of cells populated with the copied values.

Remarks

The copied data will overwrite the contents of the cells in this worksheet. You may specify the types of data copied (cell value, formulas, comments, formatting, etc.) using the CopyPasteProperties object.

Examples

C#

```
//--- Open the workbook and worksheet from which we will copy cells.
          Workbook sourceWb = xla.Open("MyDataSource.xls");
          Worksheet sourceWs = sourceWb[0];

          //--- Create the workbook and and get the worksheet
          //--- into which we will paste cells.
          Workbbok destWb = xla.Create("MyNewWorkbook");
          Worksheet destWs = destWb[0];

          //--- Define the area of cells to copy.
          Area srcArea = sourceWs.CreateArea("A1:F20");

          //--- Copy cell values, formulas and cell and number formatting.
          CopyPasteProperties copyPasteProps =
               sourceWb.CreateCopyPasteProperties(
               CopyPasteProperties.CopyPasteType.ValueFormulasAndFormatting);

          //--- Paste the data onto the destination worksheet.
          Cell whereToPaste = destSheet["A1"];
          destWs.CopyPaste(whereToPaste, srcArea, copyPasteProps);
```

vb.net

```
'--- Open the workbook and worksheet from which we will copy cells.
          Dim sourceWb As Workbook = xla.Open("MyDataSource.xls")
          Dim sourceWs As Worksheet = sourceWb(0)

          '--- Create the workbook and and get the worksheet
          '--- into which we will paste cells.
          Dim destWb As Workbook = xla.Create("MyNewWorkbook")
          Dim destWs As Worksheet = destWb(0)

          '--- Define the area of cells to copy.
          Dim srcArea As Area = sourceWs.CreateArea("A1:F20")

          '--- Copy cell values, formulas and cell and number formatting.
          Dim whereToPaste As Cell = destSheet("A1")
          Dim copyPasteProps As CopyPasteProperties = _
               sourceWb.CreateCopyPasteProperties( _
               CopyPasteProperties.CopyPasteType.ValueFormulasAndFormatting)

          '--- Paste the data onto the destination worksheet.
          destWs.CopyPaste("A1", srcArea, copyPasteProps)
```

Worksheet.CopyPaste(Cell, Area)

Description

Copies an area of cells from another Worksheet to cells in this worksheet.

Signature: C#

```
public Area CopyPaste(Cell destCell, Area sourceArea)
```

Signature: VB.NET

```
Public Function CopyPaste(ByVal destCell As Cell, ByVal sourceArea As Area) As Area
```

Parameters

destCell

A Cell indicating the upper-left cell of the destination area.

sourceArea

An area from the source workbook from which to copy.

Returns

An Area object representing the set of cells populated with the copied values.

Remarks

The copied data will overwrite the contents of the cells in this worksheet. The method behaves as if you had passed in CopyPasteProperties object initialized with CopyPasteType.All.

Examples

C#

```
//--- Open the workbook and worksheet from which we will copy cells.
          Workbook sourceWb = xla.Open("MyDataSource.xls");
          Worksheet sourceWs = sourceWb[0];

          //--- Create the workbook and and get the worksheet
          //--- into which we will paste cells.
          Workbbok destWb = xla.Create("MyNewWorkbook");
          Worksheet destWs = destWb[0];

          //--- Define the area of cells to copy.
          Area srcArea = sourceWs.CreateArea("A1:F20");

          // Paste the data onto the destination worksheet.
          // Copy all supported data types (excluding row height and column width).
          Cell whereToPaste = destSheet["A1"];
          destWs.CopyPaste(whereToPaste, srcArea);
```

vb.net

```
'--- Open the workbook and worksheet from which we will copy cells.
          Dim sourceWb As Workbook = xla.Open("MyDataSource.xls")
          Dim sourceWs As Worksheet = sourceWb(0)

          '--- Create the workbook and and get the worksheet
          '--- into which we will paste cells.
          Dim destWb As Workbook = xla.Create("MyNewWorkbook")
          Dim destWs As Worksheet = destWb(0)

          '--- Define the area of cells to copy.
          Dim srcArea As Area = sourceWs.CreateArea("A1:F20")

          '--- Paste the data onto the destination worksheet.
          '--- Copy all supported data types (excluding row height and column width).
          Dim whereToPaste As Cell = destSheet("A1")
          destWs.CopyPaste(whereToPaste, srcArea)
```

Worksheet.CopyPaste(Int32, Int32, Area, CopyPasteProperties)

Description

Copies an area of cells from another Worksheet to cells in this worksheet.

Signature: C#

```
public Area CopyPaste(int firstRow, int firstCol, Area sourceArea, CopyPasteProperties properties)
```

Signature: VB.NET

```
Public Function CopyPaste(ByVal firstRow As Integer, ByVal firstCol As Integer, ByVal sourceArea As
Area, ByVal properties As CopyPasteProperties) As Area
```

Parameters

firstRow

The zero-based index of the first row in destination area.

firstCol

The zero-based index of the first column in the destination area.

sourceArea

An area from the source workbook from which to copy.

properties

A CopyPasteProperties object that specifies the types of data to copy.

Returns

An Area object representing the set of cells populated with the copied values.

Remarks

The copied data will overwrite the contents of the cells in this worksheet. You may specify the types of data copied (cell value, formulas, comments, formatting, etc.) using the CopyPasteProperties object.

Examples

C#

```
//--- Open the workbook and worksheet from which we will copy cells.
          Workbook sourceWb = xla.Open("MyDataSource.xls");
          Worksheet sourceWs = sourceWb[0];

          //--- Create the workbook and and get the worksheet
          //--- into which we will paste cells.
          Workbbok destWb = xla.Create("MyNewWorkbook");
          Worksheet destWs = destWb[0];

          //--- Define the area of cells to copy.
          Area srcArea = sourceWs.CreateArea("A1:F20");

          //--- Copy cell values, formulas and cell and number formatting.
          CopyPasteProperties copyPasteProps =
               sourceWb.CreateCopyPasteProperties(
               CopyPasteProperties.CopyPasteType.ValueFormulasAndFormatting);

          //--- Paste the data onto the destination worksheet.
          destWs.CopyPaste(0, 0, srcArea, copyPasteProps);
```

vb.net

```
'--- Open the workbook and worksheet from which we will copy cells.
          Dim sourceWb As Workbook = xla.Open("MyDataSource.xls")
          Dim sourceWs As Worksheet = sourceWb(0)

          '--- Create the workbook and and get the worksheet
          '--- into which we will paste cells.
          Dim destWb As Workbook = xla.Create("MyNewWorkbook")
          Dim destWs As Worksheet = destWb(0)

          '--- Define the area of cells to copy.
          Dim srcArea As Area = sourceWs.CreateArea("A1:F20")

          '--- Copy cell values, formulas and cell and number formatting.
          Dim copyPasteProps As CopyPasteProperties = _
               sourceWb.CreateCopyPasteProperties( _
               CopyPasteProperties.CopyPasteType.ValueFormulasAndFormatting)

          '--- Paste the data onto the destination worksheet.
          destWs.CopyPaste(0, 0, srcArea, copyPasteProps)
```

Worksheet.CopyPaste(Int32, Int32, Area)

Description

Copies an area of cells from another Worksheet to cells in this worksheet.

Signature: C#

```
public Area CopyPaste(int firstRow, int firstCol, Area sourceArea)
```

Signature: VB.NET

```
Public Function CopyPaste(ByVal firstRow As Integer, ByVal firstCol As Integer, ByVal sourceArea As
Area) As Area
```

Parameters

firstRow

The zero-based index of the first row in destination area.

firstCol

The zero-based index of the first column in the destination area.

sourceArea

An area from the source workbook from which to copy.

Returns

An Area object representing the set of cells populated with the copied values.

Remarks

The copied data will overwrite the contents of the cells in this worksheet. The method behaves as if you had passed in CopyPasteProperties object initialized with CopyPasteType.All.

Examples

C#

```
//--- Open the workbook and worksheet from which we will copy cells.
          Workbook sourceWb = xla.Open("MyDataSource.xls");
          Worksheet sourceWs = sourceWb[0];

          //--- Create the workbook and and get the worksheet
          //--- into which we will paste cells.
          Workbbok destWb = xla.Create("MyNewWorkbook");
          Worksheet destWs = destWb[0];

          //--- Define the area of cells to copy.
          Area srcArea = sourceWs.CreateArea("A1:F20");

          //--- Paste the data onto the destination worksheet.
          //--- Copy all supported data types (excluding row height and column width).
          destWs.CopyPaste(0, 0, srcArea);
```

vb.net

```
'--- Open the workbook and worksheet from which we will copy cells.
          Dim sourceWb As Workbook = xla.Open("MyDataSource.xls")
          Dim sourceWs As Worksheet = sourceWb(0)

          '--- Create the workbook and and get the worksheet
          '--- into which we will paste cells.
          Dim destWb As Workbook = xla.Create("MyNewWorkbook")
          Dim destWs As Worksheet = destWb(0)

          '--- Define the area of cells to copy.
          Dim srcArea As Area = sourceWs.CreateArea("A1:F20")

          '--- Paste the data onto the destination worksheet.
          '--- Copy all supported data types (excluding row height and column width).
          destWs.CopyPaste(0, 0, srcArea)
```

Worksheet.CopyPaste(String, Area, CopyPasteProperties)

Description

Copies an area of cells from another Worksheet to cells in this worksheet.

Signature: C#

```
public Area CopyPaste(System.String destCell, Area sourceArea, CopyPasteProperties properties)
```

Signature: VB.NET

```
Public Function CopyPaste(ByVal destCell As String, ByVal sourceArea As Area, ByVal properties As
CopyPasteProperties) As Area
```

Parameters

destCell

A string indicating the upper-left cell of the destination area.

sourceArea

An area from the source workbook from which to copy.

properties

A CopyPasteProperties object that specifies the types of data to copy.

Returns

An Area object representing the set of cells populated with the copied values.

Remarks

The copied data will overwrite the contents of the cells in this worksheet. You may specify the types of data copied (cell value, formulas, comments, formatting, etc.) using the CopyPasteProperties object.

Examples

C#

```
//--- Open the workbook and worksheet from which we will copy cells.
          Workbook sourceWb = xla.Open("MyDataSource.xls");
          Worksheet sourceWs = sourceWb[0];

          //--- Create the workbook and and get the worksheet
          //--- into which we will paste cells.
          Workbbok destWb = xla.Create("MyNewWorkbook");
          Worksheet destWs = destWb[0];

          //--- Define the area of cells to copy.
          Area srcArea = sourceWs.CreateArea("A1:F20");

          //--- Copy cell values, formulas and cell and number formatting.
          CopyPasteProperties copyPasteProps =
               sourceWb.CreateCopyPasteProperties(
               CopyPasteProperties.CopyPasteType.ValueFormulasAndFormatting);

          //--- Paste the data onto the destination worksheet.
          destWs.CopyPaste("A1", srcArea, copyPasteProps);
```

vb.net

```
'--- Open the workbook and worksheet from which we will copy cells.
          Dim sourceWb As Workbook = xla.Open("MyDataSource.xls")
          Dim sourceWs As Worksheet = sourceWb(0)

          '--- Create the workbook and and get the worksheet
          '--- into which we will paste cells.
          Dim destWb As Workbook = xla.Create("MyNewWorkbook")
          Dim destWs As Worksheet = destWb(0)

          '--- Define the area of cells to copy.
          Dim srcArea As Area = sourceWs.CreateArea("A1:F20")

          '--- Copy cell values, formulas and cell and number formatting.
          Dim copyPasteProps As CopyPasteProperties = _
               sourceWb.CreateCopyPasteProperties( _
               CopyPasteProperties.CopyPasteType.ValueFormulasAndFormatting)

          '--- Paste the data onto the destination worksheet.
          destWs.CopyPaste("A1", srcArea, copyPasteProps)
```

Worksheet.CopyPaste(String, Area)

Description

Copies an area of cells from another Worksheet to cells in this worksheet.

Signature: C#

```
public Area CopyPaste(System.String destCell, Area sourceArea)
```

Signature: VB.NET

```
Public Function CopyPaste(ByVal destCell As String, ByVal sourceArea As Area) As Area
```

Parameters

destCell

A string indicating the upper-left cell of the destination area.

sourceArea

An area from the source workbook from which to copy.

Returns

An Area object representing the set of cells populated with the copied values.

Remarks

The copied data will overwrite the contents of the cells in this worksheet. The method behaves as if you had passed in CopyPasteProperties object initialized with CopyPasteType.All.

Examples

C#

```
//--- Open the workbook and worksheet from which we will copy cells.
          Workbook sourceWb = xla.Open("MyDataSource.xls");
          Worksheet sourceWs = sourceWb[0];

          //--- Create the workbook and and get the worksheet
          //--- into which we will paste cells.
          Workbbok destWb = xla.Create("MyNewWorkbook");
          Worksheet destWs = destWb[0];

          //--- Define the area of cells to copy.
          Area srcArea = sourceWs.CreateArea("A1:F20");

          //--- Paste the data onto the destination worksheet.
          //--- Copy all supported data types (excluding row height and column width).
          destWs.CopyPaste("A1", srcArea);
```

vb.net

```
'--- Open the workbook and worksheet from which we will copy cells.
          Dim sourceWb As Workbook = xla.Open("MyDataSource.xls")
          Dim sourceWs As Worksheet = sourceWb(0)

          '--- Create the workbook and and get the worksheet
          '--- into which we will paste cells.
          Dim destWb As Workbook = xla.Create("MyNewWorkbook")
          Dim destWs As Worksheet = destWb(0)

          '--- Define the area of cells to copy.
          Dim srcArea As Area = sourceWs.CreateArea("A1:F20")

          '--- Paste the data onto the destination worksheet.
          '--- Copy all supported data types (excluding row height and column width).
          destWs.CopyPaste("A1", srcArea)
```

Worksheet.CreateAnchor(Int32, Int32, Double, Double)

Description

Creates an anchor at a specified position in the worksheet.

Signature: C#

```
public Anchor CreateAnchor(int rowNumber, int columnNumber, double offsetX, double offsetY)
```

Signature: VB.NET

```
Public Function CreateAnchor(ByVal rowNumber As Integer, ByVal columnNumber As Integer, ByVal offsetX
As Double, ByVal offsetY As Double) As Anchor
```

Parameters

rowNumber

The 0-based row number of the cell in which the anchor will be positioned.

columnNumber

The 0-based column number of the cell in which the anchor will be positioned.

offsetX

The X offset from the cell's left side as a percentage of the cell's width. If this worksheet is a chartsheet, this value is a percentage of the chart's width. This value must be between 0 and 100.

offsetY

The Y offset from the top of the cell as a percentage of the cell's height. If this worksheet is a chartsheet, this value is a percentage of the chart's height. This value must be between 0 and 100.

Returns

An Anchor object.

Remarks

An anchor represents the position of a floating (non-cell data) object within a spreadsheet. The position is maintained as a row number + a percentage of the row's width and a column number + a percentage of the column's height.

If this Worksheet is actually a Chartsheet, the rowNumber and columnNumber values are disregarded and the offsetX and offsetY values are interperted to be a percentage of the Chart's width and height, respectivly.

Examples

C#

```
Anchor anch = ws.CreateAnchor(4, 1, 30, 50);
```

vb.net

```
Dim anch As Anchor = ws.CreateAnchor(4, 1, 30, 50)
```

Worksheet.CreateArea

Overload	Description
CreateArea(Int32, Int32, Int32, Int32)	Defines a rectangular Area of cells in the worksheet.
CreateArea(String)	Defines a rectangular Area of cells in the worksheet.

Worksheet.CreateArea(Int32, Int32, Int32, Int32)

Description

Defines a rectangular Area of cells in the worksheet.

Signature: C#

```
public Area CreateArea(int firstRow, int firstColumn, int numRows, int numColumns)
```

Signature: VB.NET

```
Public Function CreateArea(ByVal firstRow As Integer, ByVal firstColumn As Integer, ByVal numRows As
Integer, ByVal numColumns As Integer) As Area
```

Parameters

firstRow

0-based first row of the area.

firstColumn

0-based first column of the area.

numRows

Number of rows in the area. Maximum: 65536.

numColumns

Number of columns in the area. Maximum: 256.

Returns

An `Area` object.

Examples

C#

```
Area a = ws.CreateArea(4, 4, 15, 6);
```

vb.net

```
Dim a As Area = ws.CreateArea(4, 4, 15, 6)
```

Worksheet.CreateArea(String)

Description

Defines a rectangular Area of cells in the worksheet.

Signature: C#

```
public Area CreateArea(System.String areaFormula)
```

Signature: VB.NET

```
Public Function CreateArea(ByVal areaFormula As String) As Area
```

Parameters

areaFormula

A formula representing the area, e.g. "A3:B5". The formula must be local to the worksheet and should not contain a sheet reference.

Returns

An `Area` object.

Examples

C#

```
Area a = ws.CreateArea("B7:H24");
```

vb.net

```
Dim a As Area = ws.CreateArea("B7:H24")
```

Worksheet.CreateAreaOfColumns(Int32, Int32)

This method has been deprecated, as it is not the most efficient way to edit Column properties. Instead, use the Worksheet.GetColumnProperties(Int32) method to return the ColumnProperties object of the desired column.

Description

Returns an Area object representing all the cells in a specified group of columns. An area is a rectangular collection of cells.

Signature: C#

```
public Area CreateAreaOfColumns(int firstCol, int numCols)
```

Signature: VB.NET

```
Public Function CreateAreaOfColumns(ByVal firstCol As Integer, ByVal numCols As Integer) As Area
```

Parameters

firstCol

The 0-based index of the first column.

numCols

The number of columns to include.

Returns

An `Area` object representing the set of cells in the specified columns.

Examples

C#

```
Area a = ws.CreateAreaOfColumns(0, 25);
```

vb.net

```
Dim a As Area = ws.CreateAreaOfColumns(0, 25)
```

Worksheet.CreateAreaOfRows(Int32, Int32)

This method has been deprecated, as it is not the most efficient way to edit Row properties. Instead, use the Worksheet.GetRowProperties(Int32) method to return the RowProperties object of the desired row.

Description

Returns an Area object representing all the cells in a specified group of rows. An area is a rectangular collection of cells.

Signature: C#

```
public Area CreateAreaOfRows(int firstRow, int numRows)
```

Signature: VB.NET

```
Public Function CreateAreaOfRows(ByVal firstRow As Integer, ByVal numRows As Integer) As Area
```

Parameters

firstRow

The 0-based index of the first row.

numRows

The number of rows to include.

Returns

An `Area` object representing the set of cells in the specified rows.

Examples

C#

```
Area a = ws.CreateAreaOfRows(0, 25);
```

vb.net

```
Dim a As Area = ws.CreateAreaOfRows(0, 25)
```

Worksheet.CreateNamedRange

Overload	Description
CreateNamedRange(Int32, Int32, Int32, Int32, String)	Creates a named range, containing one area. A range is a collection of areas; an area is a rectangular collection of cells.
CreateNamedRange(String, String)	Creates a named range from a specified formula. A range is a collection of areas; an area is a rectangular collection of cells. This method can be used to create a non-rectangular range containing multiple rectangular areas.

Worksheet.CreateNamedRange(Int32, Int32, Int32, Int32, String)

Description

Creates a named range, containing one area. A range is a collection of areas; an area is a rectangular collection of cells.

Signature: C#

```
public Range CreateNamedRange(int firstRow, int firstColumn, int numRows, int numColumns,
System.String rangeName)
```

Signature: VB.NET

```
Public Function CreateNamedRange(ByVal firstRow As Integer, ByVal firstColumn As Integer, ByVal
numRows As Integer, ByVal numColumns As Integer, ByVal rangeName As String) As Range
```

Parameters

firstRow

0-based first row of the range.

firstColumn

0-based first column column of the range.

numRows

Number of rows in the area. Maximum: 65536.

numColumns

Number of columns in the range. Maximum: 256.

rangeName

The name of the new range.

Returns

A Range object.

Examples

C#

```
Range rng = ws.CreateNamedRange(4, 4, 15, 6, "Range1");
```

vb.net

```
Dim rng As Range = ws.CreateNamedRange(4, 4, 15, 6, "Range1")
```

Worksheet.CreateNamedRange(String, String)

Description

Creates a named range from a specified formula. A range is a collection of areas; an area is a rectangular collection of cells. This method can be used to create a non-rectangular range containing multiple rectangular areas.

Signature: C#

```
public Range CreateNamedRange(System.String rangeFormula, System.String rangeName)
```

Signature: VB.NET

```
Public Function CreateNamedRange(ByVal rangeFormula As String, ByVal rangeName As String) As Range
```

Parameters

rangeFormula

Formula representing the range, for example "=Sheet1!A1:G10". The formula must be three-dimensional (specify a sheet), and can include only sheet and cell references, and operators joining those references (range ':', intersection ' ', and union ',').

rangeName

The name of the new range.

Returns

A Range object.

Examples

C#

```
Range rng =
                ws.CreateRange("=Sheet1!$B$12:$H$21 Sheet1!$F$18:$K$29", "Range1");
```

vb.net

```
Dim rng As Range = _
                ws.CreateRange("=Sheet1!$B$12:$H$21 Sheet1!$F$18:$K$29", "Range1")
```

Worksheet.CreateRange(String)

Description

Creates a range from a specified formula. A range is a collection of areas; an area is a rectangular collection of cells. This method can be used to create a non-rectangular range containing multiple rectangular areas.

Signature: C#

```
public Range CreateRange(System.String rangeFormula)
```

Signature: VB.NET

```
Public Function CreateRange(ByVal rangeFormula As String) As Range
```

Parameters

rangeFormula

Formula representing the range, for example "=A1:G10". The formula must be two-dimensional, and can include cell references, and operators joining those references (range ':', intersection ' ', and union ','). Functions that return ranges cannot be used.

Returns

A Range object.

Examples

C#

```
Range rng = ws.CreateRange("=$B$12:$H$21 $F$18:$K$29");
```

vb.net

```
Dim rng As Range = ws.CreateRange("=$B$12:$H$21 $F$18:$K$29")
```

Worksheet.DeleteColumn(Int32)

Description

Deletes a column and its contents from the worksheet. Columns after the deleted column will be moved to the left.

Signature: C#

```
public void DeleteColumn(int columnNumber)
```

Signature: VB.NET

```
Public Sub DeleteColumn(ByVal columnNumber As Integer)
```

Parameters

columnNumber

The 0-based number of the column to remove.

Examples

C#

```
ws.DeleteColumn(9);
```

vb.net

```
ws.DeleteColumn(9)
```

Worksheet.DeleteColumns(Int32, Int32)

Description

Deletes a number of columns and its contents from the worksheet. Columns after the deleted columns will be moved to the left.

Signature: C#

```
public void DeleteColumns(int columnNumber, int columnCount)
```

Signature: VB.NET

```
Public Sub DeleteColumns(ByVal columnNumber As Integer, ByVal columnCount As Integer)
```

Parameters

columnNumber

The 0-based number of the column to remove.

columnCount

The number of columns to delete.

Examples

C#

```
ws.DeleteColumns(3,4);
```

vb.net

```
ws.DeleteColumns(3,4)
```

Worksheet.DeleteRow(Int32)

Description

Deletes a specified row and its contents from the worksheet. Rows below the deleted row will be moved up.

Signature: C#

```
public void DeleteRow(int rowNumber)
```

Signature: VB.NET

```
Public Sub DeleteRow(ByVal rowNumber As Integer)
```

Parameters

rowNumber

The 0-based number of the row to remove.

Examples

C#

```
ws.DeleteRow(70);
```

vb.net

```
ws.DeleteRow(70)
```

Worksheet.DeleteRows(Int32, Int32)

Description

Deletes a specified number of rows and its contents from the worksheet. Rows below the deleted rows will be moved up.

Signature: C#

```
public void DeleteRows(int rowNumber, int rowCount)
```

Signature: VB.NET

```
Public Sub DeleteRows(ByVal rowNumber As Integer, ByVal rowCount As Integer)
```

Parameters

rowNumber

The 0-based number of the row to remove.

rowCount

The number of the rows to remove.

Examples

C#

```
ws.DeleteRows(70, 500);
```

vb.net

```
ws.DeleteRows(70, 500)
```

Worksheet.FirstShownColumn

Description

Sets or returns the 0-based index of the first column that is shown in the worksheet.

Signature: C#

```
public int FirstShownColumn{ get; set; }
```

Signature: VB.NET

```
Public Property FirstShownColumn() As Integer
```

Examples

C#

```
//--- Get FirstShownColumn
          int firstCol = ws.FirstShownColumn;

          //--- Set FirstShownColumn
          ws.FirstShownColumn = 5;
```

vb.net

```
'--- Get FirstShownColumn
          Dim firstCol As Integer = ws.FirstShownColumn

          '--- Set FirstShownColumn
          ws.FirstShownColumn = 5
```

Worksheet.FirstShownRow

Description

Sets or returns the 0-based index of the first row that is shown in the worksheet.

Signature: C#

```
public int FirstShownRow{ get; set; }
```

Signature: VB.NET

```
Public Property FirstShownRow() As Integer
```

Examples

C#

```
//--- Get FirstShownRow
          int firstRow = ws.FirstShownRow;

          //--- Set FirstShownRow
          ws.FirstShownRow = 5;
```

vb.net

```
'--- Get FirstShownRow
          Dim firstRow As Integer = ws.FirstShownRow

          '--- Set FirstShownRow
          ws.FirstShownRow = 5
```

Worksheet.FreezePanes

Description

Freezes panes in a worksheet or returns the cell at which the panes are split. Set the property to a Cell object. The worksheet will be split above the cell, to the left of it, or both.

Signature: C#

```
public Cell FreezePanes{ get; set; }
```

Signature: VB.NET

```
Public Property FreezePanes() As Cell
```

Examples

C#

```
//--- Get FreezePanes
          Cell freezeCell = ws.FreezePanes;

          //--- Set FreezePanes
          ws.FreezePanes = ws.Cells["C5"];
```

vb.net

```
'--- Get FreezePanes
          Dim freezeCell As Cell = ws.FreezePanes

          '--- Set FreezePanes
          ws.FreezePanes = ws.Cells("C5");
```

Worksheet.GetColumnProperties(Int32)

Description

Returns a ColumnProperties object representing the column specified by index.

Signature: C#

```
public ColumnProperties GetColumnProperties(int columnIndex)
```

Signature: VB.NET

```
Public Function GetColumnProperties(ByVal columnIndex As Integer) As ColumnProperties
```

Parameters

columnIndex

The 0-based index of the column.

Returns

A `ColumnProperties` object representing the specified column.

Examples

C#

```
ColumnProperties colProps = ws.GetColumnProperties(0);
```

vb.net

```
Dim colProps As ColumnProperties = ws.GetColumnProperties(0)
```

Worksheet.GetNamedObject(String)

Description

Returns the NamedObject object that represents the name of a specified array, number, picture, or range. If the named object does not exist, the method returns null.

Signature: C#

```
public NamedObject GetNamedObject(System.String name)
```

Signature: VB.NET

```
Public Function GetNamedObject(ByVal name As String) As NamedObject
```

Parameters

name

A string representation of the object's name.

Returns

A `NamedObject` object that represents the name of a specified array, number, picture, or range. If the named object does not exist, the method returns null.

Examples

C#

```
NamedObject obj = ws.GetNamedObject("MyNamedObject");
```

vb.net

```
Dim obj As NamedObject = ws.GetNamedObject("MyNamedObject")
```

Worksheet.GetNamedRange(String)

Description

Returns the named Range object that specified by name. If the named range does not exist, the method returns null.

Signature: C#

```
public Range GetNamedRange(System.String name)
```

Signature: VB.NET

```
Public Function GetNamedRange(ByVal name As String) As Range
```

Parameters

name

The name of the range.

Returns

A named `Range` object, or null if the range does not exist.

Examples

C#

```
Range named = ws.GetNamedRange("MyNamedRange");
```

vb.net

```
Dim named As Range = ws.GetNamedRange("MyNamedRange")
```

Worksheet.GetRowProperties(Int32)

Description

Returns a RowProperties object representing the row specified by index.

Signature: C#

```
public RowProperties GetRowProperties(int rowIndex)
```

Signature: VB.NET

```
Public Function GetRowProperties(ByVal rowIndex As Integer) As RowProperties
```

Parameters

rowIndex

The 0-based index of the row.

Returns

A `RowProperties` object representing the specified row.

Examples

C#

```
RowProperties rowProps = ws.GetRowProperties(0);
```

vb.net

```
Dim rowProps As RowProperties = ws.GetRowProperties(0)
```

Worksheet.GridlinesColor

Description

Sets or returns a `Color` object representing the color of the gridlines in the worksheet.

Signature: C#

```
public Color GridlinesColor{ get; set; }
```

Signature: VB.NET

```
Public Property GridlinesColor() As Color
```

Examples

C#

```
//--- Get GridlinesColor
          Color gridColor = ws.GridlinesColor;

          //--- Set GridlinesColor
          ws.GridlinesColor = wb.Palette.GetClosestColor(System.Drawing.Color.Blue);
```

vb.net

```
'--- Get GridlinesColor
          Dim gridColor As Color = ws.GridlinesColor

          '--- Set GridlinesColor
          ws.GridlinesColor = wb.Palette.GetClosestColor(System.Drawing.Color.Blue)
```

Worksheet.GroupColumns(Int32, Int32, Boolean)

Description

Groups or outlines a contiguous set of columns.

Signature: C#

```
public void GroupColumns(int startColumnIndex, int numColumns, boolean collapsed)
```

Signature: VB.NET

```
Public Sub GroupColumns(ByVal startColumnIndex As Integer, ByVal numColumns As Integer, ByVal
collapsed As Boolean)
```

Parameters

startColumnIndex

The 0-based index of the starting column.

numColumns

The number of Columns to group.

collapsed

Whether or not the group is collapsed.

Examples

C#

```
ws.GroupColumns(0, 5, true);
```

vb.net

```
ws.GroupColumns(0, 5, True)
```

Worksheet.GroupRows(Int32, Int32, Boolean)

Description

Groups or outlines a contiguous set of rows.

Signature: C#

```
public void GroupRows(int startRowIndex, int numRows, boolean collapsed)
```

Signature: VB.NET

```
Public Sub GroupRows(ByVal startRowIndex As Integer, ByVal numRows As Integer, ByVal collapsed As
Boolean)
```

Parameters

startRowIndex

The 0-based index of the starting row.

numRows

The number of rows to group.

collapsed

Whether or not the group is collapsed.

Examples

C#

```
ws.GroupRows(0, 15, true);
```

vb.net

```
ws.GroupRows(0, 15, True)
```

Worksheet.Hyperlinks

Description

Returns the collection of hyperlinks in the worksheet.

Signature: C#

```
public Hyperlinks Hyperlinks{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Hyperlinks() As Hyperlinks
```

Examples

C#

```
Hyperlinks links = ws.Hyperlinks;
```

vb.net

```
Dim links As Hyperlinks = ws.Hyperlinks
```

Worksheet.ImportData

Overload	Description
ImportData(Object()(), Cell)	Imports data from a two-dimensional array of objects to cells in the worksheet.
ImportData(Object()(), String(), Cell, DataImportProperties)	Imports data from a two-dimensional array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object(,), Cell)	Imports data from a rectangular array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Object(,), String(), Cell, DataImportProperties)	Imports data from a rectangular array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.IDataReader, Cell)	Imports data from an `IDataReader` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.IDataReader, Cell, DataImportProperties)	Imports data from an `IDataReader` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataTable, Cell)	Imports data from an ADO.NET `DataTable` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataTable, Cell, DataImportProperties)	Imports data from an ADO.NET `DataTable` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataView, Cell)	Imports data from an ADO.NET `DataView` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(System.Data.DataView, Cell, DataImportProperties)	Imports data from an ADO.NET `DataView` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

SharePoint Integration Extension Methods

Overload	Description
ImportData(Cell, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint List to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Cell, Microsoft.SharePoint.SPList)	Imports data from a SharePoint List to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint View to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Imports data from a SharePoint View to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Worksheet.ImportData(Object(,), Cell)

Description

Imports data from a rectangular array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Object[,] data, Cell cell)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object(,), ByVal cell As Cell) As Area
```

Parameters

data

A rectangular array of values to import to the worksheet. The first dimension corresponds to row and the second to column.
Thus, an array of data { {"A","X"},{"B","Y"},{"C","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

startCell

The cell at which to start entering the imported values.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataArray, cel);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataArray, cel)
```

Worksheet.ImportData(Object(,), String(), Cell, DataImportProperties)

Description

Imports data from a rectangular array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Object[,] data, System.String[] columnNames, Cell cell,
DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object(,), ByVal columnNames As String(), ByVal cell As Cell,
ByVal props As DataImportProperties) As Area
```

Parameters

data

A rectangular array of values to import to the worksheet. The first dimension corresponds to row and the second to column.
Thus, an array of data { {"A","X"},{"B","Y"},{"C","Z"} } would be inserted into the worksheet as:

A	X

B	Y
C	Z

If you enable DataImportProperties.Transpose , the format will be [row][column], so:
{ {"A","B","C"},{"X","Y","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

columnNames

An array of column names for the imported data. `columnNames` and the column dimension of `data` must contain the same number of elements.

startCell

The cell at which to start entering the imported values.

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataArray, fieldNames, cel, importProps);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataArray, fieldNames, cel, importProps)
```

Worksheet.ImportData(Object()(), Cell)

Description

Imports data from a two-dimensional array of objects to cells in the worksheet.

Signature: C#

```
public Area ImportData(System.Object[][] data, Cell startCell)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object()(), ByVal startCell As Cell) As Area
```

Parameters

data

A two-dimensional array of values to import to the worksheet.
The first dimension corresponds to column and the second to row (that is, Object[column][row]).
Thus, an array of data

```
{{"A","X"},{"B","Y"},{"C","Z"}}
```

would be inserted into the worksheet as:

A	X
B	Y
C	Z

startCell

The cell at which to start entering the imported values.

Returns

An Area object representing the set of cells populated with the imported values.

Remarks

The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Examples

C#

```
Area importedArea = ws.ImportData(dataArray, cel);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataArray, cel)
```

Worksheet.ImportData(Object()(), String(), Cell, DataImportProperties)

Description

Imports data from a two-dimensional array of objects to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Object[][] data, System.String[] columnNames, Cell startCell,
DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As Object()(), ByVal columnNames As String(), ByVal startCell As
Cell, ByVal props As DataImportProperties) As Area
```

Parameters

data

A two-dimensional array of values to import to the worksheet.
By default, the first dimension corresponds to column and the second to row (that is, Object[column][row]).
Thus, an array of data { {"A","X"},{"B","Y"},{"C","Z"} } would be inserted into the worksheet as:

A	X
B	Y
C	Z

If you enable DataImportProperties.Transpose , the format will be [row][column], so:
{ {"A","B","C"},{"X","Y","Z"} } would be inserted into the worksheet as:

A	X
B	Y

C Z

columnNames

An array of column names for the imported data. `columnNames` and the column dimension of `data` must contain the same number of elements.

startCell

The cell at which to start entering the imported values.

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataArray, fieldNames, cel, importProps);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataArray, fieldNames, cel, importProps)
```

Worksheet.ImportData(System.Data.DataTable, Cell, DataImportProperties)

Description

Imports data from an ADO.NET `DataTable` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.DataTable data, Cell cell, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataTable, ByVal cell As Cell, ByVal props As
DataImportProperties) As Area
```

Parameters

data

The `DataTable` to import.

startCell

The cell at which to start entering the imported values.

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataDT, cel, importProps);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataDT, cel, importProps)
```

Worksheet.ImportData(System.Data.DataTable, Cell)

Description

Imports data from an ADO.NET `DataTable` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.DataTable data, Cell cell)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataTable, ByVal cell As Cell) As Area
```

Parameters

data

The `DataTable` to import.

startCell

The cell at which to start entering the imported values.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataDT, cel);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataDT, cel)
```

Worksheet.ImportData(System.Data.DataView, Cell, DataImportProperties)

Description

Imports data from an ADO.NET `DataView` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.DataView data, Cell cell, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataView, ByVal cell As Cell, ByVal props As
DataImportProperties) As Area
```

Parameters

data

The `DataView` to import.

startCell

The cell at which to start entering the imported values.

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataDV, cel, importProps);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataDV, cel, importProps)
```

Worksheet.ImportData(System.Data.DataView, Cell)

Description

Imports data from an ADO.NET `DataView` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.DataView data, Cell cell)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.DataView, ByVal cell As Cell) As Area
```

Parameters

data

The `DataView` to import.

startCell

The cell at which to start entering the imported values.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataDV, cel);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataDV, cel)
```

Worksheet.ImportData(System.Data.IDataReader, Cell, DataImportProperties)

Description

Imports data from an `IDataReader` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.IDataReader data, Cell cell, DataImportProperties props)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.IDataReader, ByVal cell As Cell, ByVal props As
DataImportProperties) As Area
```

Parameters

data

An instance of `System.Data.SqlClient.SqlDataReader`, `System.Data.OleDb.OleDbDataReader` or

```
Microsoft.AnalysisServices.AdomdClient.AdomdDataReader
```

.
startCell

The cell at which to start entering the imported values.

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataDR, cel, importProps);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataDR, cel, importProps)
```

Worksheet.ImportData(System.Data.IDataReader, Cell)

Description

Imports data from an `IDataReader` to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but

existing formatting will be preserved.

Signature: C#

```
public Area ImportData(System.Data.IDataReader data, Cell cell)
```

Signature: VB.NET

```
Public Function ImportData(ByVal data As System.Data.IDataReader, ByVal cell As Cell) As Area
```

Parameters

data

An instance of `System.Data.SqlClient.SqlDataReader`, `System.Data.OleDb.OleDbDataReader` or

```
Microsoft.AnalysisServices.AdomdClient.AdomdDataReader
```

.

startCell

The cell at which to start entering the imported values.

Returns

An Area object representing the set of cells populated with the imported values.

Examples

C#

```
Area importedArea = ws.ImportData(dataDR, cel);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(dataDR, cel)
```

Worksheet.InsertColumn

Overload	Description
InsertColumn(Int32)	Inserts a column in the worksheet to the left of the specified column.
InsertColumn(Int32, ColumnInsertBehavior)	Inserts a column in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.
InsertColumn(Int32, ColumnInsertBehavior, ColumnInsertBehavior)	Inserts a column in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.

Worksheet.InsertColumn(Int32, Style.ColumnInsertBehavior, Style.ColumnInsertBehavior)

Description

Inserts a column in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.

Signature: C#

```
public void InsertColumn(int columnNumber, ColumnInsertBehavior columnCellsCopyBehavior,
ColumnInsertBehavior columnCopyBehavior)
```

Signature: VB.NET

```
Public Sub InsertColumn(ByVal columnNumber As Integer, ByVal columnCellsCopyBehavior As
ColumnInsertBehavior, ByVal columnCopyBehavior As ColumnInsertBehavior)
```

Parameters

columnNumber

The 0-based number of the column to insert.

columnCellsCopyBehavior

The Style.ColumnInsertBehavior that defines the source style for each cell in the new column. Each cell in the new column will be given the style of its neighbor identified in this parameter.

columnCopyBehavior

The Style.ColumnInsertBehavior that defines how the default column style will be set for the new column. This style is cells before the 'columnCellsCopyBehavior', so the 'columnCellsCopyBehavior' can be used to override the column-wide style.

Examples

C#

```
ws.InsertColumn(3, Style.InsertBehavior.CopyFromLeft);
```

vb.net

```
ws.InsertColumn(3, Style.InsertBehavior.CopyFromLeft)
```

Worksheet.InsertColumn(Int32, Style.ColumnInsertBehavior)

Description

Inserts a column in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.

Signature: C#

```
public void InsertColumn(int columnNumber, ColumnInsertBehavior columnCellsCopyBehavior)
```

Signature: VB.NET

```
Public Sub InsertColumn(ByVal columnNumber As Integer, ByVal columnCellsCopyBehavior As
ColumnInsertBehavior)
```

Parameters

columnNumber

The 0-based number of the column to insert.

columnCellsCopyBehavior

The Style.ColumnInsertBehavior that defines the source style for each cell in the new column. Each cell in the new column will be given the style of its neighbor identified in this parameter.

Examples

C#

```
ws.InsertColumn(3, Style.InsertBehavior.CopyFromLeft);
```

vb.net

```
ws.InsertColumn(3, Style.InsertBehavior.CopyFromLeft)
```

Worksheet.InsertColumn(Int32)

Description

Inserts a column in the worksheet to the left of the specified column.

Signature: C#

```
public void InsertColumn(int columnNumber)
```

Signature: VB.NET

```
Public Sub InsertColumn(ByVal columnNumber As Integer)
```

Parameters

columnNumber

The 0-based number of the column to insert.

Examples

C#

```
ws.InsertColumn(3);
```

vb.net

```
ws.InsertColumn(3)
```

Worksheet.InsertColumns

Overload	Description
InsertColumns(Int32, Int32)	Inserts a block of columns in the worksheet to the left of the specified column.
InsertColumns(Int32, Int32, ColumnInsertBehavior)	Inserts a block of columns in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.
InsertColumns(Int32, Int32, ColumnInsertBehavior, ColumnInsertBehavior)	Inserts a block of columns in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.

Worksheet.InsertColumns(Int32, Int32, Style.ColumnInsertBehavior, Style.ColumnInsertBehavior)

Description

Inserts a block of columns in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.

Signature: C#

```
public void InsertColumns(int columnNumber, int columnCount, ColumnInsertBehavior
columnCellsCopyBehavior, ColumnInsertBehavior columnCopyBehavior)
```

Signature: VB.NET

```
Public Sub InsertColumns(ByVal columnNumber As Integer, ByVal columnCount As Integer, ByVal
columnCellsCopyBehavior As ColumnInsertBehavior, ByVal columnCopyBehavior As ColumnInsertBehavior)
```

Parameters

columnNumber

The 0-based number of the column to insert.

columnCount

The number of columns to insert.

columnCellsCopyBehavior

The Style.ColumnInsertBehavior that defines the source style for each cell in the new column. Each cell in the new column will be given the style of its neighbor identified in this parameter.

columnCopyBehavior

The Style.ColumnInsertBehavior that defines how the default column style will be set for the new column. This style is cells before the 'columnCellsCopyBehavior', so the 'columnCellsCopyBehavior' can be used to override the column-wide style.

Examples

C#

```
ws.InsertColumn(3, 10, Style.InsertBehavior.CopyFromLeft);
```

vb.net

```
ws.InsertColumn(3, 10, Style.InsertBehavior.CopyFromLeft)
```

Worksheet.InsertColumns(Int32, Int32, Style.ColumnInsertBehavior)

Description

Inserts a block of columns in the worksheet to the left of the specified column, copying the style based on the `copyBehavior` parameter.

Signature: C#

```
public void InsertColumns(int columnNumber, int columnCount, ColumnInsertBehavior
columnCellsCopyBehavior)
```

Signature: VB.NET

```
Public Sub InsertColumns(ByVal columnNumber As Integer, ByVal columnCount As Integer, ByVal
columnCellsCopyBehavior As ColumnInsertBehavior)
```

Parameters

columnNumber

The 0-based number of the column to insert.

columnCount

The number of columns to insert.

columnCellsCopyBehavior

The Style.ColumnInsertBehavior that defines the source style for each cell in the new column. Each cell in the new column will be given the style of its neighbor identified in this parameter.

Examples

C#

```
ws.InsertColumn(3, 10, Style.InsertBehavior.CopyFromLeft);
```

vb.net

```
ws.InsertColumn(3, 10, Style.InsertBehavior.CopyFromLeft)
```

Worksheet.InsertColumns(Int32, Int32)

Description

Inserts a block of columns in the worksheet to the left of the specified column.

Signature: C#

```
public void InsertColumns(int columnNumber, int columnCount)
```

Signature: VB.NET

```
Public Sub InsertColumns(ByVal columnNumber As Integer, ByVal columnCount As Integer)
```

Parameters

columnNumber

The 0-based number of the column to insert.

columnCount

The number of columns to insert.

Examples

C#

```
ws.InsertColumns(3, 10);
```

vb.net

```
ws.InsertColumns(3, 10)
```

Worksheet.InsertHorizontalPageBreak(Cell)

Description

Inserts a horizontal page break in the worksheet after the specified cell.

Signature: C#

```
public void InsertHorizontalPageBreak(Cell afterCell)
```

Signature: VB.NET

```
Public Sub InsertHorizontalPageBreak(ByVal afterCell As Cell)
```

Parameters

afterCell

A Cell object representing the cell at which to insert a horizontal page break.

Examples

C#

```
ws.InsertHorizontalPageBreak(ws.Cells["A25"]);
```

vb.net

```
ws.InsertHorizontalPageBreak(ws.Cells("A25"))
```

Worksheet.InsertRow

Overload	Description
InsertRow(Int32)	Inserts a row in the worksheet above the specified row.
InsertRow(Int32, RowInsertBehavior)	Inserts a row in the worksheet above the specified row.
InsertRow(Int32, RowInsertBehavior, Style.RowInsertBehavior)	Inserts a row in the worksheet above the specified row.

Worksheet.InsertRow(Int32, Style.RowInsertBehavior, Style.RowInsertBehavior)

Description

Inserts a row in the worksheet above the specified row.

Signature: C#

```
public void InsertRow(int rowNumber, RowInsertBehavior rowCellsCopyBehavior, RowInsertBehavior
rowCopyBehavior)
```

Signature: VB.NET

```
Public Sub InsertRow(ByVal rowNumber As Integer, ByVal rowCellsCopyBehavior As RowInsertBehavior,
ByVal rowCopyBehavior As RowInsertBehavior)
```

Parameters

rowNumber

The 0-based number of the row to insert.

rowCellsCopyBehavior

The Style.RowInsertBehavior that defines the source style for each cell in the new row. Each cell in the new row will be given the style of its neighbor identified in this parameter.

rowCopyBehavior

The Style.RowInsertBehavior that defines how the default row style will be set for the new row. This style is cells before the 'rowCellsCopyBehavior', so the 'rowCellsCopyBehavior' can be used to override the row-wide style.

Examples

C#

```
ws.InsertRow(3,
                Style.InsertBehavior.CopyAreaFromBelow,
                Style.RowInsertBehavior.Ignore);
```

vb.net

```
ws.InsertRow(3, _
               Style.InsertBehavior.CopyAreaFromBelow, _
               Style.RowInsertBehavior.Ignore)
```

Worksheet.InsertRow(Int32, Style.RowInsertBehavior)

Description

Inserts a row in the worksheet above the specified row.

Signature: C#

```
public void InsertRow(int rowNumber, RowInsertBehavior rowCellsCopyBehavior)
```

Signature: VB.NET

```
Public Sub InsertRow(ByVal rowNumber As Integer, ByVal rowCellsCopyBehavior As RowInsertBehavior)
```

Parameters

rowNumber

The 0-based number of the row to insert.

rowCellsCopyBehavior

The Style.RowInsertBehavior that defines the source style for each cell in the new row. Each cell in the new row will be given the style of its neighbor identified in this parameter.

Examples

C#

```
ws.InsertRow(3, Style.InsertBehavior.CopyAreaFromBelow);
```

vb.net

```
ws.InsertRow(3, Style.InsertBehavior.CopyAreaFromBelow)
```

Worksheet.InsertRow(Int32)

Description

Inserts a row in the worksheet above the specified row.

Signature: C#

```
public void InsertRow(int rowNumber)
```

Signature: VB.NET

```
Public Sub InsertRow(ByVal rowNumber As Integer)
```

Parameters

rowNumber

The 0-based number of the row to insert.

Examples

C#

```
ws.InsertRow(3);
```

vb.net

```
ws.InsertRow(3)
```

Worksheet.InsertRows

Overload	Description
InsertRows(Int32, Int32)	Inserts a number of rows in the worksheet above the specified row.
InsertRows(Int32, Int32, RowInsertBehavior)	Inserts a number of rows in the worksheet above the specified row.
InsertRows(Int32, Int32, RowInsertBehavior, RowInsertBehavior)	Inserts a number of rows in the worksheet above the specified row.

Worksheet.InsertRows(Int32, Int32, Style.RowInsertBehavior, Style.RowInsertBehavior)

Description

Inserts a number of rows in the worksheet above the specified row.

Signature: C#

```
public void InsertRows(int rowNumber, int rowCount, RowInsertBehavior rowCellsCopyBehavior,
RowInsertBehavior rowCopyBehavior)
```

Signature: VB.NET

```
Public Sub InsertRows(ByVal rowNumber As Integer, ByVal rowCount As Integer, ByVal
rowCellsCopyBehavior As RowInsertBehavior, ByVal rowCopyBehavior As RowInsertBehavior)
```

Parameters

rowNumber

The 0-based number of the row to insert.

rowCount

The number of rows to insert.

rowCellsCopyBehavior

The Style.RowInsertBehavior that defines the source style for each cell in the new row. Each cell in the new row will be given the style of its neighbor identified in this parameter.

rowCopyBehavior

The Style.RowInsertBehavior that defines how the default row style will be set for the new row. This style is cells before the 'rowCellsCopyBehavior', so the 'rowCellsCopyBehavior' can be used to override the row-wide style.

Examples

C#

```
ws.InsertRows(3,
              100,
              Style.InsertBehavior.CopyAreaFromBelow,
              Style.RowInsertBehavior.Ignore);
```

vb.net

```
ws.InsertRows(3, _
               100, _
               Style.InsertBehavior.CopyAreaFromBelow, _
               Style.RowInsertBehavior.Ignore)
```

Worksheet.InsertRows(Int32, Int32, Style.RowInsertBehavior)

Description

Inserts a number of rows in the worksheet above the specified row.

Signature: C#

```
public void InsertRows(int rowNumber, int rowCount, RowInsertBehavior rowCellsCopyBehavior)
```

Signature: VB.NET

```
Public Sub InsertRows(ByVal rowNumber As Integer, ByVal rowCount As Integer, ByVal
rowCellsCopyBehavior As RowInsertBehavior)
```

Parameters

rowNumber

The 0-based number of the row to insert.

rowCount

The number of rows to insert.

rowCellsCopyBehavior

The Style.RowInsertBehavior that defines the source style for each cell in the new row. Each cell in the new row will be given the style of its neighbor identified in this parameter.

Examples

C#

```
ws.InsertRows(3, 100, Style.InsertBehavior.CopyAreaFromBelow);
```

vb.net

```
ws.InsertRows(3, 100, Style.InsertBehavior.CopyAreaFromBelow)
```

Worksheet.InsertRows(Int32, Int32)

Description

Inserts a number of rows in the worksheet above the specified row.

Signature: C#

```
public void InsertRows(int rowNumber, int rowCount)
```

Signature: VB.NET

```
Public Sub InsertRows(ByVal rowNumber As Integer, ByVal rowCount As Integer)
```

Parameters

rowNumber

The 0-based number of the row to insert.

rowCount

The number of rows to insert.

Examples

C#

```
ws.InsertRows(3, 100);
```

vb.net

```
ws.InsertRows(3, 100)
```

Worksheet.InsertVerticalPageBreak(Cell)

Description

Inserts a vertical page break in the worksheet to right of the specified cell.

Signature: C#

```
public void InsertVerticalPageBreak(Cell rightOfCell)
```

Signature: VB.NET

```
Public Sub InsertVerticalPageBreak(ByVal rightOfCell As Cell)
```

Parameters

rightOfCell

A Cell object representing the cell at which to insert a vertical page break.

Examples

C#

```
ws.InsertVerticalPageBreak(ws.Cells["G1"]);
```

vb.net

```
ws.InsertVerticalPageBreak(ws.Cells("G1"))
```

Worksheet.IsProtected

Description

Returns whether or not the worksheet is write-protected.

Signature: C#

```
public boolean IsProtected{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property IsProtected() As Boolean
```

Examples

C#

```
bool writeProtected = ws.IsProtected;
```

vb.net

```
Dim writeProtected As Boolean = ws.IsProtected
```

Worksheet.IsSelected

Description

Returns whether this worksheet is selected. Multiple worksheets can be selected.

Signature: C#

```
public boolean IsSelected{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property IsSelected() As Boolean
```

Examples

C#

```
bool selected = ws.IsSelected;
```

vb.net

```
Dim selected As Boolean = ws.IsSelected
```

Worksheet.Item

Overload	Description
Item(Int32, Int32)	Returns the cell at a specified row and column position. This property is an indexer for the Worksheet class.
Item(String)	Returns the cell at the specified Excel-style reference. This property is an indexer for the Worksheet class.

Worksheet.Item(Int32, Int32)

Description

Returns the cell at a specified row and column position. This property is an indexer for the Worksheet class.

Signature: C#

```
public Cell this[int rowNumber, int columnNumber] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal rowNumber As Integer, ByVal columnNumber As Integer) As
Cell
```

Parameters

rowNumber

The 0-based row number of the cell to retrieve.

columnNumber

The 0-based column number of the cell to retrieve.

Returns

A Cell object representing the cell at the specified position.

Examples

C#

```
//--- Get Cell by index
          Cell cellA1 = ws[0,0];
```

vb.net

```
'--- Get Cell by index
          Dim cellA1 As Cell = ws(0,0)
```

Worksheet.Item(String)

Description

Returns the cell at the specified Excel-style reference. This property is an indexer for the Worksheet class.

Signature: C#

```
public Cell this[string reference] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal reference As String) As Cell
```

Parameters

rowNumber

The Excel-style reference of the cell to retrieve.

columnNumber

The Excel-style reference of the cell to retrieve.

Returns

A Cell object representing the cell at the specified Excel-style reference.

Examples

C#

```
//--- Get Cell by Excel-style reference
          Cell cellA1 = ws["A1"];
```

vb.net

```
'--- Get Cell by Excel-style reference
          Dim cellA1 As Cell = ws("A1")
```

Worksheet.Name

Description

Sets or returns the name of the worksheet. Sheet names are limited to 255 characters and must not contain the following characters: \ / ? [ ] : '

Signature: C#

```
public System.String Name{ get; set; }
```

Signature: VB.NET

```
Public Property Name() As String
```

Examples

C#

```
//--- Get SheetName
          string sheetName = ws.Name;

          //--- Set SheetName
          ws.Name = "SheetOne";
```

vb.net

```
'--- Get SheetName
          Dim sheetName As String = ws.Name

          '--- Set SheetName
          ws.Name = "SheetOne"
```

Worksheet.NamedRanges

Description

Returns the named ranges of the Worksheet as an array of NamedObject objects as an `IEnumerable` collection. Though the named ranges are not returned in any particular order, they iterable.

Signature: C#

```
public System.Collections.Generic.IEnumerable<NamedObject> NamedRanges{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property NamedRanges() As System.Collections.Generic.IEnumerable(Of NamedObject)
```

Examples

C#

```
NamedObject[] ranges = ws.NamedRanges;
```

vb.net

```
Dim ranges as NamedObject() = ws.NamedRanges
```

Worksheet.PageSetup

Description

Returns a PageSetup object representing page layout properties of a printed worksheet.

Signature: C#

```
public PageSetup PageSetup{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PageSetup() As PageSetup
```

Examples

C#

```
PageSetup ps = ws.PageSetup;
```

vb.net

```
Dim ps As PageSetup = ws.PageSetup
```

Worksheet.Pictures

Description

Returns a Pictures collection containing all pictures in the worksheet. Use this collection to add, access, and remove pictures from the worksheet.

Signature: C#

```
public Pictures Pictures{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Pictures() As Pictures
```

Examples

C#

```
Pictures allPictures = ws.Pictures;
```

vb.net

```
Dim allPictures = ws.Pictures
```

Worksheet.PopulatedCells

Description

Returns a rectanglular area of cells from the first (top, left) populated cell to the last (bottom, right) populated cell in the worksheet. The area returned will include cells that contain no data but have been formatted.

Signature: C#

```
public Area PopulatedCells{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property PopulatedCells() As Area
```

Examples

C#

```
Area a = ws.PopulatedCells;
```

vb.net

```
Dim a As Area = ws.PopulatedCells
```

Worksheet.Position

Description

Returns the 0-based index of the worksheet in the collection of all sheets in the workbook.

Signature: C#

```
public int Position{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Position() As Integer
```

Examples

C#

```
int sheetPosition = ws.Position;
```

vb.net

```
Dim sheetPosition As Integer = ws.Position
```

Worksheet.Protect(String)

Description

Write-protects the worksheet. A user will not be able to modify the worksheet in Excel without entering the specified password. This method does not encrypt the worksheet.

Signature: C#

```
public void Protect(System.String password)
```

Signature: VB.NET

```
Public Sub Protect(ByVal password As String)
```

Parameters

password

The password to use to protect the worksheet. If the password is "" or null, the worksheet will be protected without a password.

Remarks

To further protect your spreadsheet, you can protect the Workbook's structure using the Workbook.Protect() method or password encrypt the file using the Workbook.EncryptPassword property.

Examples

C#

```
ws.Protect("MyPassword");
```

vb.net

```
ws.Protect("MyPassword")
```

Worksheet.ProtectPasswordHash

Description

Sets or returns the password hash that is used to write-protect (not encrypt) the worksheet.

Signature: C#

```
public int ProtectPasswordHash{ get; set; }
```

Signature: VB.NET

```
Public Property ProtectPasswordHash() As Integer
```

Examples

C#

```
//--- Get ProtectPasswordHash
          int passwordHash = ws.ProtectPasswordHash;

          //--- Set ProtectPasswordHash
          ws.ProtectPasswordHash = 12345;
```

vb.net

```
'--- Get ProtectPasswordHash
          Dim passwordHash = ws.ProtectPasswordHash

          '--- Set ProtectPasswordHash
          ws.ProtectPasswordHash = 12345
```

Worksheet.Select()

Description

Selects the current worksheet and deselects all others. To select multiple worksheets, use Worksheets.Select().

Signature: C#

```
public void Select()
```

Signature: VB.NET

```
Public Sub [Select]()
```

Examples

C#

```
ws.Select();
```

vb.net

```
ws.Select()
```

Worksheet.ShapeGroups

Description

Returns a Groups collection containing all shape groups within the worksheet.

Signature: C#

```
public Groups ShapeGroups{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property ShapeGroups() As Groups
```

Examples

C#

```
Groups allGroups = ws.Groups;
```

vb.net

```
Dim allGroups As Groups = ws.Groups
```

Worksheet.Shapes

Description

Returns a Shapes collection containing all shapes within the worksheet.

Signature: C#

```
public Shapes Shapes{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Shapes() As Shapes
```

Examples

C#

```
Shapes allShapes = ws.Shapes;
```

vb.net

```
Dim allShapes As Shapes = ws.Shapes
```

Worksheet.SheetViewState

Description

A SheetViewState value specifies the view state of a worksheet in Excel.

Signature: C#

```
public sealed class SheetViewState : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class SheetViewState
  Inherits TypesafeEnum
```

Values

Name	Description
Normal	Normal
PageBreak	Page break

Worksheet.SheetVisibility

Description

A SheetVisibility value specifies the visibility level of a worksheet.

Signature: C#

```
public sealed class SheetVisibility : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class SheetVisibility
  Inherits TypesafeEnum
```

Values

Name	Description
Hidden	The worksheet will be hidden when the workbook is opened in Excel.
VeryHidden	The worksheet will be hidden and cannot be made visible from the Excel user interface.
Visible	The worksheet will be shown when the workbook is opened in Excel.

Worksheet.ShowFormulas

Description

Sets or returns whether formulas or their calculated values will be displayed in the worksheet. Calculated values are displayed by default.

Signature: C#

```
public boolean ShowFormulas{ get; set; }
```

Signature: VB.NET

```
Public Property ShowFormulas() As Boolean
```

Examples

C#

```
//--- Get ShowFormulas
          bool displayFormulas = ws.ShowFormulas;

          //--- Set ShowFormulas
          ws.ShowFormulas = true;
```

vb.net

```
'--- Get ShowFormulas
          Dim displayFormulas As Boolean = ws.ShowFormulas

          '--- Set ShowFormulas
          ws.ShowFormulas = True
```

Worksheet.ShowGridlines

Description

Sets or returns whether gridlines should be shown in the worksheet. Gridlines are shown by default.

Signature: C#

```
public boolean ShowGridlines{ get; set; }
```

Signature: VB.NET

```
Public Property ShowGridlines() As Boolean
```

Examples

C#

```
//--- Get ShowGridlines
          bool showGrid = ws.ShowGridlines;

          //--- Set ShowGridlines
          ws.ShowGridlines = false;
```

vb.net

```
'--- Get ShowGridlines
          Dim showGrid As Boolean = ws.ShowGridlines

          '--- Set ShowGridlines
          ws.ShowGridlines = False
```

Worksheet.ShowRowColHeaders

Description

Sets or returns whether column and row headers will be displayed in the worksheet. The headers are displayed by default.

Signature: C#

```
public boolean ShowRowColHeaders{ get; set; }
```

Signature: VB.NET

```
Public Property ShowRowColHeaders() As Boolean
```

Examples

C#

```
//--- Get ShowRowColHeaders
          bool showHeaders = ws.ShowRowColHeaders;

          //--- Set ShowRowColHeaders
          ws.ShowRowColHeaders = false;
```

vb.net

```
'--- Get ShowRowColHeaders
          Dim showHeaders As Boolean = ws.ShowRowColHeaders

          '--- Set ShowRowColHeaders
          ws.ShowRowColHeaders = False
```

Worksheet.ShowZeroValues

Description

Sets or returns whether zero values in cells will be displayed in the worksheet. Zero values are displayed by default.

Signature: C#

```
public boolean ShowZeroValues{ get; set; }
```

Signature: VB.NET

```
Public Property ShowZeroValues() As Boolean
```

Examples

C#

```
//--- Get ShowZeroValues
          bool showZeroes = ws.ShowZeroValues;

          //--- Set ShowZeroValues
          ws.ShowZeroValues = true;
```

vb.net

```
'--- Get ShowZeroValues
          Dim showZeros As Boolean = ws.ShowZeroValues

          '--- Set ShowZeroValues
          ws.ShowZeroValues = True
```

Worksheet.StandardHeight

Description

Sets or returns the height of the "Standard Row" in points (1/72 of an inch). This must be a value that when converted to character widths is between 0 - 409.

Signature: C#

```
public double StandardHeight{ get; set; }
```

Signature: VB.NET

```
Public Property StandardHeight() As Double
```

Examples

C#

```
//--- Get StandardHeight
          ws.StandardHeight = 35;

          //--- Set StandardHeight
          double rowHeight = ws.StandardHeight;
```

vb.net

```
'--- Get StandardHeight
          ws.StandardHeight = 35

          '--- Set StandardHeight
          Dim rowHeight As Double = ws.StandardHeight
```

Worksheet.StandardWidth

Description

Sets or returns the width of the "Standard Column", in points (1/72 of an inch). This must be a value that when converted to character widths is between 0 - 255.

Signature: C#

```
public double StandardWidth{ get; set; }
```

Signature: VB.NET

```
Public Property StandardWidth() As Double
```

Examples

C#

```
//--- Get StandardWidth
          ws.StandardWidth = 100;

          //--- Set StandardWidth
          double colWidth = ws.StandardWidth;
```

vb.net

```
'--- Get StandardWidth
          ws.StandardWidth = 100

          '--- Set StandardWidth
          Dim colWidth As Double = ws.StandardWidth
```

Worksheet.StandardWidthInChars

Description

Sets or returns the width of the "Standard Column" as a number of '0' characters in the font of the 'Normal' style. This must be a value between 0 - 255.

Signature: C#

```
public double StandardWidthInChars{ get; set; }
```

Signature: VB.NET

```
Public Property StandardWidthInChars() As Double
```

Examples

C#

```
//--- Get StandardWidthInChars
          ws.StandardWidthInChars = 20;

          //--- Set StandardWidthInChars
          double colWidth = ws.StandardWidthInChars;
```

vb.net

```
'--- Get StandardWidthInChars
          ws.StandardWidthInChars = 20

          '--- Set StandardWidthInChars
          Dim colWidth As Double = ws.StandardWidthInChars
```

Worksheet.SummaryColumns

Description

When grouping columns, SummaryColumns determines where the summary columns for the groups are located. By default the summary rows are located to the right of the groupings.

Signature: C#

```
public SummaryColumnsLocation SummaryColumns{ get; set; }
```

Signature: VB.NET

```
Public Property SummaryColumns() As SummaryColumnsLocation
```

Examples

C#

```
//--- Get SummaryColumns
          Worksheet.SummaryColumnsLocation location = ws.SummaryColumns;

          //--- Set SummaryColumns
          ws.SummaryColumns = Worksheet.SummaryColumnsLocation.Left;
```

vb.net

```
'--- Get SummaryColumns
          Dim location As Worksheet.SummaryColumnsLocation = ws.SummaryColumns

          '--- Set SummaryColumns
          ws.SummaryColumns = Worksheet.SummaryColumnsLocation.Left
```

Worksheet.SummaryColumnsLocation

Description

A SummaryColumnsLocation value specifies the location of the summary column for a set of grouped columns.

Signature: C#

```
public sealed class SummaryColumnsLocation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class SummaryColumnsLocation
  Inherits TypesafeEnum
```

Values

Name	Description
Left	To the left of the grouped columns
Right	To the right of the grouped columns

Worksheet.SummaryRows

Description

When grouping rows, SummaryRows determines where the summary rows for the groups are located. By default the summary rows are located below the groupings.

Signature: C#

```
public SummaryRowsLocation SummaryRows{ get; set; }
```

Signature: VB.NET

```
Public Property SummaryRows() As SummaryRowsLocation
```

Examples

C#

```
//--- Get SummaryRows
          Worksheet.SummaryRowsLocation location = ws.SummaryRows;

          //--- Set SummaryRows
          ws.SummaryRows = Worksheet.SummaryLocation.Above;
```

vb.net

```
'--- Get SummaryRows
          Dim location As Worksheet.SummaryRowsLocation = ws.SummaryRows

          '--- Set SummaryRows
          ws.SummaryRows = Worksheet.SummaryLocation.Above
```

Worksheet.SummaryRowsLocation

Description

A SummaryColumnsLocation value specifies the location of the summary row for a set of grouped rows.

Signature: C#

```
public sealed class SummaryRowsLocation : TypesafeEnum
```

Signature: VB.NET

```
Public NotInheritable Class SummaryRowsLocation
  Inherits TypesafeEnum
```

Values

Name	Description
Above	Above the grouped rows
Below	Below the grouped rows

Worksheet.TabColor

Description

Sets or returns the color of the worksheet's sheet name tab.

Signature: C#

```
public Color TabColor{ get; set; }
```

Signature: VB.NET

```
Public Property TabColor() As Color
```

Examples

C#

```
//--- Set TabColor
          ws.TabColor = wb.Palette.GetClosestColor(System.Drawing.Color.Red);

          //--- Get TabColor
          Color tabClr = ws.TabColor;
```

vb.net

```
'--- Set TabColor
          ws.TabColor = wb.Palette.GetClosestColor(System.Drawing.Color.Red)

          '--- Get TabColor
          Dim tabClr As Color = ws.TabColor
```

Worksheet.UngroupColumns(Int32, Int32)

Description

Ungroups a contiguous set of columns.

Signature: C#

```
public void UngroupColumns(int startColumnIndex, int numColumns)
```

Signature: VB.NET

```
Public Sub UngroupColumns(ByVal startColumnIndex As Integer, ByVal numColumns As Integer)
```

Parameters

startColumnIndex

The 0-based index of the starting column.

numColumns

The number of Columns to ungroup.

Examples

C#

```
ws.UngroupColumns(0, 5);
```

vb.net

```
ws.UngroupColumns(0, 5)
```

Worksheet.UngroupRows(Int32, Int32)

Description

Ungroups a contiguous set of rows.

Signature: C#

```
public void UngroupRows(int startRowIndex, int numRows)
```

Signature: VB.NET

```
Public Sub UngroupRows(ByVal startRowIndex As Integer, ByVal numRows As Integer)
```

Parameters

startRowIndex

The 0-based index of the starting row.

numRows

The number of rows to ungroup.

Examples

C#

```
ws.UngroupRows(0, 15);
```

vb.net

```
ws.UngroupRows(0, 15)
```

Worksheet.Unprotect()

Description

Removes the write-protection from the worksheet.

Signature: C#

```
public void Unprotect()
```

Signature: VB.NET

```
Public Sub Unprotect()
```

Examples

C#

```
ws.Unprotect();
```

vb.net

```
ws.Unprotect()
```

Worksheet.ViewState

Description

Sets or returns the worksheet's viewing mode, which may be either normal or page break mode.

Signature: C#

```
public SheetViewState ViewState{ get; set; }
```

Signature: VB.NET

```
Public Property ViewState() As SheetViewState
```

Examples

C#

```
//--- Get ViewState
          Worksheet.SheetViewState state = ws.ViewState;

          //--- Set ViewState
          ws.ViewState = Worksheet.SheetViewState.PageBreak;
```

vb.net

```
'--- Get ViewState
          Dim state As Worksheet.SheetViewState = ws.ViewState

          '--- Set ViewState
          ws.ViewState = Worksheet.SheetViewState.PageBreak
```

Worksheet.Visibility

Description

Sets or returns the visibility setting of the worksheet.

Signature: C#

```
public SheetVisibility Visibility{ get; set; }
```

Signature: VB.NET

```
Public Property Visibility() As SheetVisibility
```

Examples

C#

```
//--- Get Visibility
          Worksheet.Visibility vis = ws.Visibility;

          //--- Set Visibility
          ws.Visibility = Worksheet.SheetVisibility.Hidden;
```

vb.net

```
'--- Get Visibility
          Dim vis As Worksheet.Visibility = ws.Visibility

          '--- Set Visibility
          ws.Visibility = Worksheet.SheetVisibility.Hidden
```

Worksheet.Workbook

Description

Returns the Workbook object in which this worksheet resides.

Signature: C#

```
public Workbook Workbook{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Workbook() As Workbook
```

Examples

C#

```
Workbook wkbk = sheet.Worksheet;
```

vb.net

```
Dim wkbk as Workbook = sheet.Workbook
```

Worksheet.ZoomPercentage

Description

Sets or returns the percentage of standard size by which the worksheet will be magnified or reduced. Set this property to a value between 10 and 400.

Signature: C#

```
public int ZoomPercentage{ get; set; }
```

Signature: VB.NET

```
Public Property ZoomPercentage() As Integer
```

Examples

C#

```
//--- Get Zoom
          int zoom = ws.ZoomPercentage;

          //--- Set Zoom
          ws.ZoomPercentage = 50;
```

vb.net

```
'--- Get Zoom
          Dim zoom As Integer = ws.ZoomPercentage

          '--- Set Zoom
          ws.ZoomPercentage = 50
```

Worksheets

Description

The Worksheets class represents the set of worksheets in an Excel workbook. This object is used to manage (add, copy, remove, etc.) the worksheets in the workbook To return a `Worksheets` object, use Workbook.Worksheets.

Signature: C#

```
[DefaultMember("Item")]
 public sealed class Worksheets : System.Collections.Generic.IEnumerable<Worksheet>
```

Signature: VB.NET

```
<DefaultMember("Item")> _
 Public NotInheritable Class Worksheets
  Implements System.Collections.Generic.IEnumerable(Of Worksheet)
```

Properties

Name	Description
Count	Returns the number of worksheets in the workbook.

Indexers

Name	Description
Item(Int32)	Returns the worksheet at the specified position. This property is an indexer for the Worksheets class.
Item(String)	Returns the worksheet with the specified name. This property is an indexer for the Worksheets class.

Methods

Name	Description
CopySheet(Worksheet, Int32, String)	Copies a worksheet in the workbook to another position in the workbook.
CreateChartsheet(ChartType, String)	Creates a new chartsheet after the last existing worksheet in this workbook.
CreateChartsheet(ChartType, String, Int32)	Creates a new Chartsheet at a specified position in the workbook.
CreateWorksheet(String)	Creates a new worksheet after the last existing worksheet in this workbook.
CreateWorksheet(String, Int32)	Creates a new worksheet at a specified position in the workbook.
Delete(Int32)	Deletes a worksheet from the workbook.
Delete(String)	Deletes a worksheet from the workbook.
GetEnumerator()	Returns an IEnumerator for the individual Worksheets in a Worksheets object
InsertSheet(String, Int32)	Inserts a new worksheet into the workbook at a specified position.
MoveSheet(Worksheet, Int32)	Moves a worksheet to a different position in the workbook.
Select(Object())	Set the sheets that will be selected when the workbook is opened in Excel.

Worksheets.CopySheet(Worksheet, Int32, String)

Description

Copies a worksheet in the workbook to another position in the workbook.

Signature: C#

```
public void CopySheet(Worksheet sheet, int position, System.String newName)
```

Signature: VB.NET

```
Public Sub CopySheet(ByVal sheet As Worksheet, ByVal position As Integer, ByVal newName As String)
```

Parameters

sheet

A Worksheet object representing the sheet to copy.

position

The 0-based position at which to insert a copy of the specified worksheet. If `position` is less than 0, it is inserted before the first sheet in the workbook. If `position` is equal to or greater than Worksheets.Count, it is inserted after the last worksheet.

newName

the new name to use for this worksheet.

Remarks

This method is intended to allow for copying of sheets from one position to another within a single workbook. To copy content from a worksheet in one workbook to worksheet in another workbook, use the Worksheet.CopyPaste

Examples

C#

```
sheets.CopySheet(ws1, 0, "Sheet2");
```

vb.net

```
sheets.CopySheet(ws1, 0, "Sheet2")
```

Worksheets.Count

Description

Returns the number of worksheets in the workbook.

Signature: C#

```
public int Count{ get; }
```

Signature: VB.NET

```
Public ReadOnly Property Count() As Integer
```

Examples

C#

```
int numSheets = oWorksheets.Count;
```

vb.net

```
Dim numSheets As Integer = oWorksheets.Count
```

Worksheets.CreateChartsheet

Overload	Description
CreateChartsheet(ChartType, String)	Creates a new chartsheet after the last existing worksheet in this workbook.
CreateChartsheet(ChartType, String, Int32)	Creates a new Chartsheet at a specified position in the workbook.

Worksheets.CreateChartsheet(ChartType, String, Int32)

Description

Creates a new Chartsheet at a specified position in the workbook.

Signature: C#

```
public Chartsheet CreateChartsheet(ChartType type, System.String name, int position)
```

Signature: VB.NET

```
Public Function CreateChartsheet(ByVal type As ChartType, ByVal name As String, ByVal position As
Integer) As Chartsheet
```

Parameters

type

The type of chart to create. Specify both a chart type and sub-type, for example, ChartType.Bar.Clustered.

name

Name for the new chartsheet.

position

The 0-based position at which to insert the chartsheet. If `position` is greater than Worksheets.Count, an exception will be thrown.

Returns

A Chartsheet object representing the chartsheet created.

Examples

C#

```
ChartSheet cs =
     sheets.CreateWorksheet(ChartType.Bar.Clustered, "NewSheet", 0);
```

vb.net

```
Dim cs As ChartSheet = _
     sheets.CreateWorksheet(ChartType.Bar.Clustered, "NewSheet", 0)
```

Worksheets.CreateChartsheet(ChartType, String)

Description

Creates a new chartsheet after the last existing worksheet in this workbook.

Signature: C#

```
public Chartsheet CreateChartsheet(ChartType type, System.String name)
```

Signature: VB.NET

```
Public Function CreateChartsheet(ByVal type As ChartType, ByVal name As String) As Chartsheet
```

Parameters

type

The type of chart to create. Specify both a chart type and sub-type, for example, ChartType.Bar.Clustered.

name

Name for the new chartsheet.

Returns

A Chartsheet object representing the chartsheet created.

Examples

C#

```
ChartSheet cs =
     sheets.CreateChartsheet(ChartType.Bar.Clustered, "NewSheet");
```

vb.net

```
Dim cs As ChartSheet = _
     sheets.CreateChartsheet(ChartType.Bar.Clustered, "NewSheet")
```

Worksheets.CreateWorksheet

Overload	Description
CreateWorksheet(String)	Creates a new worksheet after the last existing worksheet in this workbook.
CreateWorksheet(String, Int32)	Creates a new worksheet at a specified position in the workbook.

Worksheets.CreateWorksheet(String, Int32)

Description

Creates a new worksheet at a specified position in the workbook.

Signature: C#

```
public Worksheet CreateWorksheet(System.String name, int position)
```

Signature: VB.NET

```
Public Function CreateWorksheet(ByVal name As String, ByVal position As Integer) As Worksheet
```

Parameters

name

Name for the new worksheet.

position

The 0-based position at which to insert the worksheet. If `position` is greater than Worksheets.Count, an exception will be thrown.

Returns

A Worksheet object representing the worksheet created.

Examples

C#

```
Worksheet ws = sheets.CreateWorksheet("NewSheet", 0);
```

vb.net

```
Dim ws As Worksheet = sheets.CreateWorksheet("NewSheet", 0)
```

Worksheets.CreateWorksheet(String)

Description

Creates a new worksheet after the last existing worksheet in this workbook.

Signature: C#

```
public Worksheet CreateWorksheet(System.String name)
```

Signature: VB.NET

```
Public Function CreateWorksheet(ByVal name As String) As Worksheet
```

Parameters

name

Name for the new worksheet.

Returns

A Worksheet object representing the worksheet created.

Examples

C#

```
Worksheet ws = sheets.CreateWorksheet("NewSheet");
```

vb.net

```
Dim ws As Worksheet = sheets.CreateWorksheet("NewSheet")
```

Worksheets.Delete

Overload	Description
Delete(Int32)	Deletes a worksheet from the workbook.
Delete(String)	Deletes a worksheet from the workbook.

Worksheets.Delete(Int32)

Description

Deletes a worksheet from the workbook.

Signature: C#

```
public void Delete(int position)
```

Signature: VB.NET

```
Public Sub Delete(ByVal position As Integer)
```

Parameters

position

The 0-based sheet position.

Examples

C#
`sheets.Delete(0);`

vb.net
`sheets.Delete(0)`

Worksheets.Delete(String)

Description

Deletes a worksheet from the workbook.

Signature: C#

```
public void Delete(System.String name)
```

Signature: VB.NET

```
Public Sub Delete(ByVal name As String)
```

Parameters

name

The name of the sheet to remove.

Examples

C#
`sheets.Delete("Sheet3");`

vb.net
`sheets.Delete("Sheet3")`

Worksheets.GetEnumerator()

Description

Returns an IEnumerator for the individual Worksheets in a Worksheets object

Signature: C#

```
public IEnumerator GetEnumerator()
```

Signature: VB.NET

```
Public Function GetEnumerator As IEnumerator
```

Returns

An enumerator for the individual Worksheets in a Worksheets object as a System.Collections.IEnumerator

Examples

C#

```
IEnumerator enumerator = worksheets.GetEnumerator();
```

vb.net

```
Dim enumerator as IEnumerator = worksheets.GetEnumerator()
```

Worksheets.InsertSheet(String, Int32)

Description

Inserts a new worksheet into the workbook at a specified position.

Signature: C#

```
public Worksheet InsertSheet(System.String name, int position)
```

Signature: VB.NET

```
Public Function InsertSheet(ByVal name As String, ByVal position As Integer) As Worksheet
```

Parameters

name

The name of the sheet to create and insert.

position

The 0-based position at which to insert the worksheet. If `position` is less than 0, it is inserted before the first sheet in the workbook. If `position` is equal to or greater than Worksheets.Count, it is inserted after the last worksheet.

Returns

A Worksheet object representing the inserted worksheet.

Examples

C#

```
sheets.InsertSheet("NewSheet", 1);
```

vb.net

```
sheets.InsertSheet("NewSheet", 1)
```

Worksheets.Item

Overload	Description
Item(Int32)	Returns the worksheet at the specified position. This property is an indexer for the Worksheets class.
Item(String)	Returns the worksheet with the specified name. This property is an indexer for the Worksheets class.

Worksheets.Item(Int32)

Description

Returns the worksheet at the specified position. This property is an indexer for the Worksheets class.

Signature: C#

```
public Worksheet this[int position] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal position As Integer) As Worksheet
```

Parameters

position

The 0-based position the worksheet to get.

Returns

The worksheet at the specified position.

Examples

C#

```
Worksheet Sheet1 = oWorkbook.Worksheet[0];
```

vb.net

```
Dim Sheet1 As Worksheet = oWorkbook.Worksheet(0)
```

Worksheets.Item(String)

Description

Returns the worksheet with the specified name. This property is an indexer for the Worksheets class.

Signature: C#

```
public Worksheet this[string name] { get; }
```

Signature: VB.NET

```
Public Default ReadOnly Property Item(ByVal name As String) As Worksheet
```

Parameters

name

The name of the worksheet to get.

Returns

The worksheet with the specified name.

Examples

C#

```
Worksheet Sheet1 = oWorkbook.Worksheet["Sheet1"];
```

vb.net

```
Dim Sheet1 As Worksheet = oWorkbook.Worksheet("Sheet1")
```

Worksheets.MoveSheet(Worksheet, Int32)

Description

Moves a worksheet to a different position in the workbook.

Signature: C#

```
public void MoveSheet(Worksheet sheet, int position)
```

Signature: VB.NET

```
Public Sub MoveSheet(ByVal sheet As Worksheet, ByVal position As Integer)
```

Parameters

sheet

A Worksheet object representing the sheet to move.

position

The 0-based position to which to move the specified worksheet. If `position` is less than 0, the sheet will be the first sheet in the workbook. If `position` is equal to or greater than Worksheets.Count, it will be inserted after the last worksheet.

Examples

C#

```
sheets.MoveSheet(oWorkSheet, 3);
```

vb.net

```
sheets.MoveSheet(oWorkSheet, 3)
```

Worksheets.Select(Object())

Description

Set the sheets that will be selected when the workbook is opened in Excel.

Signature: C#

```
public void Select(System.Object[] sheets)
```

Signature: VB.NET

```
Public Sub [Select](ByVal sheets As Object())
```

Parameters

sheets

Array of sheet indexes or sheet names.

Examples

C#

```
sheets.Select(arrSheets);
```

vb.net

```
sheets.Select(arrSheets)
```

ExcelTemplate Programmer's Reference

- DataBindingProperties — The DataBindingProperties object contains the properties information necessary to bind data to an ExcelWriter template.
- ExcelTemplate — The ExcelTemplate object represents an ExcelWriter template spreadsheet.

DataBindingProperties

Description

The DataBindingProperties object contains the properties information necessary to bind data to an ExcelWriter template.

Signature: C#

```
public sealed class DataBindingProperties
```

Signature: VB.NET

```
Public NotInheritable Class DataBindingProperties
```

Remarks

The MaxRows, Transpose, and WorksheetName properties, which had been set as parameters in the SetDataSource methods of ExcelTemplate, are now bundled into the `DataBindingProperties` object and can be applied to multiple discrete data binding calls in a single application.

The `DataBindingProperties` object must be created using the ExcelTemplate.CreateDataBindingProperties method:

C#

```
DataBindingProperties dbp = xlt.CreateDataBindingProperties();
```

vb.net

```
Dim dbp As DataBindingProperties = xlt.CreateDataBindingProperties()
```

Properties

Name	Description

MaxRows	Sets or returns the maximum number of rows to be bound to the template file when the ExcelTemplate.BindData or ExcelTemplate.BindColumnData methods are called.
Transpose	Sets or returns whether the data being bound to a template file should be transposed before being bound to the template using the ExcelTemplate.BindData method.
WorksheetName	Sets or returns the name of the worksheet to which the data will be bound during an ExcelTemplate.BindData, ExcelTemplate.BindCellData, ExcelTemplate.BindColumnData, or ExcelTemplate.BindRowData method call.

DataBindingProperties.MaxRows

Description

Sets or returns the maximum number of rows to be bound to the template file when the ExcelTemplate.BindData or ExcelTemplate.BindColumnData methods are called.

Signature: C#

```
public int MaxRows{ get; set; }
```

Signature: VB.NET

```
Public Property MaxRows() As Integer
```

Remarks

Any rows in the data source after the MaxRows value will not be added to the template.

`MaxRows` is a **Read/Write** property.

Examples

C#

```
DataBindingProperties dbp = xlt.CreateDataBindingProperties();
          dbp.MaxRows = 20;
          xlt.BindData(twodim, names, "TwoDimArray", dbp);
          xlt.Save(Page.Response, "output.xls", false);
```

vb.net

```
Dim dbp As DataBindingProperties = xlt.CreateDataBindingProperties()
          dbp.MaxRows = 20
          xlt.BindData(twodim, names, "TwoDimArray", dbp)
          xlt.Save(Page.Response, "output.xls", False)
```

DataBindingProperties.Transpose

Description

Sets or returns whether the data being bound to a template file should be transposed before being bound to the template using the ExcelTemplate.BindData method.

Signature: C#

```
public boolean Transpose{ get; set; }
```

Signature: VB.NET

```
Public Property Transpose() As Boolean
```

Remarks

If this property is set to `true`, the rows and columns of the data source will be flipped so that the first row of data is inserted in the first column, the second row of data into the second column, and so on. The default value for this property is `false`.

Transpose is a **Read/Write** property.

Examples

C#

```
DataBindingProperties dbp = xlt.CreateDataBindingProperties();
          dbp.Transpose = true;
          xlt.BindData(twodim, names, "TwoDimArray", dbp);
          xlt.Save(Page.Response, "output.xls", false);
```

vb.net

```
Dim dbp As DataBindingProperties = xlt.CreateDataBindingProperties()
          dbp.Transpose = False
          xlt.BindData(twodim, names, "TwoDimArray", dbp)
          xlt.Save(Page.Response, "output.xls", False)
```

DataBindingProperties.WorksheetName

Description

Sets or returns the name of the worksheet to which the data will be bound during an ExcelTemplate.BindData, ExcelTemplate.BindCellData, ExcelTemplate.BindColumnData, or ExcelTemplate.BindRowData method call.

Signature: C#

```
public System.String WorksheetName{ get; set; }
```

Signature: VB.NET

```
Public Property WorksheetName() As String
```

Remarks

By specifying the worksheet name, you can import from different data sources for the same data marker in different worksheets within the same workbook. The default value for WorksheetName is `null` (`Nothing` in VB.NET), meaning that the data source will be applied to the data marker for the entire workbook.

`WorksheetName` is a **Read/Write** property.

Examples

C#

```
DataBindingProperties dbp = xlt.CreateDataBindingProperties();
          dbp.WorksheetName = "first worksheet";
          xlt.BindData(twodim, names, "TwoDimArray", dbp);
          xlt.Save(Page.Response, "output.xls", false);
```

vb.net

```
Dim dbp As DataBindingProperties = xlt.CreateDataBindingProperties()
          dbp.WorksheetName = "first worksheet"
          xlt.BindData(twodim, names, "TwoDimArray", dbp)
          xlt.Save(Page.Response, "output.xls", False)
```

ExcelTemplate

Description

The ExcelTemplate object represents an ExcelWriter template spreadsheet.

Signature: C#

```
public class ExcelTemplate
```

Signature: VB.NET

```
Public Class ExcelTemplate
```

Remarks

An ExcelWriter template is a file created in Microsoft Excel that contains data markers.

A data marker specifies a database column, variable, or array to insert in the spreadsheet column containing the marker.

The `ExcelTemplate` object is in the `SoftArtisans.OfficeWriter.ExcelWriter` namespace. The object can be referenced as `SoftArtisans.OfficeWriter.ExcelWriter.ExcelTemplate`. To minimize typing and errors, use an `Import` directive to import the namespace to the aspx page, and reference the object as `ExcelTemplate`, without the namespace prefix.

Examples

If you are coding directly in the .aspx page, following the Page directive, include:

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

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.ExcelWriter;
```

vb.net

```
Imports SoftArtisans.OfficeWriter.ExcelWriter
```

To create an instance of the `ExcelTemplate` object use:

C#

```
ExcelTemplate oExcelTemplate = new ExcelTemplate();
```

vb.net

```
Dim oExcelTemplate As New ExcelTemplate()
```

Fields

ALL_ROWS	Used as DataBindingProperties.MaxRows, to indicate that as much of the data source should be imported as possible.

Properties

Name	Description
ContentType	Sets or returns the generated spreadsheet's MIME content type.
DecryptPassword	To decrypt an encrypted template, set DecryptPassword to the password set using EncryptPassword or in Microsoft Excel (Tools -> Options... -> Security).
EncryptPassword	If you set EncryptPassword, ExcelWriter will encrypt the generated spreadsheet - when Process is called - using RC4 encryption.
ExcludeMacros	Sets whether ExcelWriter should remove template macros from the generated spreadsheet or include them.
LicenseKey	Sets or returns the 23-character license key that enables ExcelWriter for the current instance.
PreserveStrings	If you set PreserveStrings to `true`, ExcelWriter will preserve numeric strings as strings.
RemoveExtraDataMarkers	Sets whether ExcelWriter should remove data markers in the template that do not bind to data sources in code.
StretchCellReferencesInCharts	Sets or returns whether to stretch cell references in charts.
Version	Returns the exact version of ExcelWriter.

Methods

Name	Description
BindCellData(Object, String, DataBindingProperties)	Sets an object as a data source for a single cell in a template.
BindColumnData(Object(), String, DataBindingProperties)	Sets an array of objects as a data source for a single column in the template.
BindColumnData(System.Collections.IDictionary, String, DataBindingProperties)	Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.
BindColumnData(System.Collections.IEnumerable, String, DataBindingProperties)	Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.
BindData(Object()(), String(), String, DataBindingProperties)	Sets a two-dimensional - possibly jagged - array of objects as a template data source.
BindData(Object(,), String(), String, DataBindingProperties)	Sets a rectangular array of objects as a template data source.
BindData(System.Data.IDataReader, String, DataBindingProperties)	Sets an ADO.NET IDataReader as a data source to bind to template data markers.
BindData(System.Data.DataSet, String, DataBindingProperties)	Sets an ADO.NET DataSet as a data source to bind to template data markers.
BindData(System.Data.DataTable, String, DataBindingProperties)	Sets an ADO.NET DataTable as a data source to bind to template data markers.
BindRowData(Object(), String(), String, DataBindingProperties)	Sets an array of objects as a data source for a single row in a template.
BindRowData(System.Collections.IDictionary, String, DataBindingProperties)	Sets a row's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.
BindRowData(System.Collections.IEnumerable, String(), String, DataBindingProperties)	Sets a row's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.
BindRowData(System.Data.DataSet, String, DataBindingProperties)	Sets a DataSet as a data source to bind to a row in the template.
BindRowData(System.Data.DataTable, String, DataBindingProperties)	Sets a DataTable as a data source to bind to a row in the template.
BindRowData(System.Data.DataView, String, DataBindingProperties)	Sets a DataView as a data source to bind to a row in the template.
CreateDataBindingProperties()	Creates a DataBindingProperties object for assigning the property values associated with one or more data binding calls.

ExcelTemplate()	Creates a new ExcelTemplate object.
Open(ExcelApplication, Workbook)	Passes a spreadsheet from ExcelApplication to ExcelTemplate.
Open(String)	Opens an ExcelWriter template spreadsheet. A template is a file created in Microsoft Excel that contains data markers where data source values will be inserted.
Open(System.IO.Stream)	Opens an ExcelWriter template from a `System.IO.Stream`. An ExcelWriter template is a file created in Microsoft Excel that contains data markers where data source values will be inserted.
Process()	The Process method enters data source values in a template's data markers, and creates the output file (the new spreadsheet) in memory. The Save method can then save the output file to disk, stream it to the browser, or both.
Save(String)	Saves the generated Excel file on the server. The ASPNET account (IIS5), NETWORK SERVICE account (IIS6 or IIS7), or the authenticated user must have Write access to the destination directory specified by the `outputFileName` parameter.
Save(System.IO.Stream)	Generates an Excel binary file and streams it to the specified `System.IO.Stream` or, a class derived from `System.IO.Stream` (for example, `System.IO.FileStream`). If you pass Save a `System.IO.FileStream`, ExcelWriter will save the generated file on the server. If you pass `Save Response.OutputStream`, ExcelWriter will stream the the generated file to the client.
Save(System.Web.HttpResponse)	If you pass Save an `HttpResponse` object object, ExcelWriter will stream the generated file to the client.
Save(System.Web.HttpResponse, String, Boolean)	If you pass Save an HttpResponse object, ExcelWriter 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 Excel.
SetCellDataSource(Object, String)	Sets an object as a data source for a single cell in a template.
SetColumnDataSource(Object(), String)	Sets an array of objects as a data source for a single column in the template. You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, ExcelTemplate.SetRowDataSource, and ExcelTemplate.SetDataSource.
SetColumnDataSource(Object(), String, Int32)	Sets an array of objects as a data source for a single column in the template. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.
SetColumnDataSource(System.Collections.IDictionary, String)	Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.
SetColumnDataSource(System.Collections.IDictionary, String, Int32)	Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.
SetColumnDataSource(System.Collections.IEnumerable, String)	Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.
SetColumnDataSource(System.Collections.IEnumerable, String, Int32)	Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.
SetDataSource(Object()(), String(), String)	Sets a two-dimensional - possibly jagged - array of objects as a template data source.
SetDataSource(Object()(), String(), String, Int32, Boolean)	**(Deprecated)** Sets a two-dimensional - possibly jagged - array of objects as a template data source.
SetDataSource(Object(,), String(), String)	Sets a rectangular array of objects as a template data source.
SetDataSource(Object(,), String(), String, Int32, Boolean)	Sets a rectangular array of objects as a template data source.
SetDataSource(System.Data.IDataReader, String)	Sets an ADO.NET IDataReader as a data source to bind to template data markers.
SetDataSource(System.Data.IDataReader, String, Int32)	Sets an IDataReader as a data source to bind to template data markers.

SetDataSource(System.Data.DataSet, String)	Sets an ADO.NET DataSet as a data source to bind to template data markers.
SetDataSource(System.Data.DataSet, String, Int32)	Sets an ADO.NET DataSet as a data source to bind to template data markers.
SetDataSource(System.Data.DataTable, String)	Sets an ADO.NET DataTable as a data source to bind to template data markers.
SetDataSource(System.Data.DataTable, String, Int32)	Sets an ADO.NET DataTable as a data source to bind to template data markers.
SetRowDataSource(Object(), String(), String)	Sets an array of objects as a data source for a single row in a template.
SetRowDataSource(System.Collections.IDictionary, String)	Sets a row's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.
SetRowDataSource(System.Collections.IEnumerable, String(), String)	Sets a row's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.
SetRowDataSource(System.Data.DataSet, String)	Sets a DataSet as a data source to bind to a row in the template. Only the first row of the DataSet will be used. This method is commonly used to create a set of key-value pairs via a single row of data.
SetRowDataSource(System.Data.DataTable, String)	Sets a DataTable as a data source to bind to a row in the template. Only the first row of the DataTable will be used. This method is commonly used to create a set of key-value pairs via a single row of data.
SetRowDataSource(System.Data.DataView, String)	Sets a DataView as a data source to bind to a row in the template. Only the first row of the DataView will be used. This method is commonly used to create a set of key-value pairs via a single row of data.

Extension Methods

Introduced in build 7.5

Name	Description
BindData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint View as a template data source.
BindData(Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint List as a template data source.
BindRowData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint View as a data source for a single row in a template.
BindRowData(Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint List as a data source for a single row in a template.
Open(Microsoft.SharePoint.SPDocumentLibrary, String)	Opens a spreadsheet from a SharePoint Document Library.
Open(Microsoft.SharePoint.SPListItem, String)	Opens a spreadsheet from a SharePoint List Item.
Save(Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Generates an Excel binary or OOXML file and saves it to a SharePoint Document Library.
Save(Microsoft.SharePoint.SPListItem, String)	Generates an Excel binary or OOXML file and saves it as an attachment to a SharePoint list item.

ExcelTemplate.ALL_ROWS

Description

Used as DataBindingProperties.MaxRows, to indicate that as much of the data source should be imported as possible.

Signature: C#

```
public static const int ALL_ROWS = 1048576;
```

Signature: VB.NET

```
Public Shared Const ALLROWS As Integer = 1048576
```

ExcelTemplate.BindCellData(Object, String, DataBindingProperties)

Description

Sets an object as a data source for a single cell in a template.

Signature: C#

```
public virtual void BindCellData(System.Object cellData, System.String dataSourceName,
DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindCellData(ByVal cellData As Object, ByVal dataSourceName As String, ByVal
[property] As DataBindingProperties)
```

Parameters

cellData

An object to use as the data source.

dataMarkerName

The name of the template data marker at which to insert the data source value. This parameter must be specified, but can be null or an empty string when setting the first data source to bind to the template.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the `DataBindingProperties.MaxRows`, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindCellData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
//--- Set a cell data source for the data marker
          //--- %%=$Company
          xlt.BindCellData("SoftArtisans",
               "Company",
               xlt.CreateDataBindingProperties());
```

vb.net

```
'--- Set a cell data source for the data marker
          '--- %%=$Company
          xlt.BindCellData("SoftArtisans", _
               "Company", _
               xlt.CreateDataBindingProperties())
```

ExcelTemplate.BindColumnData

Overload	Description
BindColumnData(Object(), String, DataBindingProperties)	Sets an array of objects as a data source for a single column in the template.
BindColumnData(System.Collections.IDictionary, String, DataBindingProperties)	Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.
BindColumnData(System.Collections.IEnumerable, String, DataBindingProperties)	Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.

ExcelTemplate.BindColumnData(Object(), String, DataBindingProperties)

Description

Sets an array of objects as a data source for a single column in the template.

Signature: C#

```
public virtual void BindColumnData(System.Object[] columnData, System.String dataSourceName,
DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindColumnData(ByVal columnData As Object(), ByVal dataSourceName As String,
ByVal [property] As DataBindingProperties)
```

Parameters

columnData

An array of objects to use as the data source.

dataSourceName

The name of the template data marker at which to insert the values imported from the data source.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindColumnData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$SimpleArray
          string[] onedim = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"};
          xlt.BindColumnData(onedim,
               "SimpleArray",
               xlt.CreateDataBindingProperties());
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$SimpleArray
          Dim onedim As String() = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"}
          xlt.BindColumnData(onedim, _
               "SimpleArray", _
               xlt.CreateDataBindingProperties())
```

ExcelTemplate.BindColumnData(System.Collections.IDictionary, String, DataBindingProperties)

Description

Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.

Signature: C#

```
public virtual void BindColumnData(System.Collections.IDictionary dataSource, System.String
dataSourceName, DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindColumnData(ByVal dataSource As System.Collections.IDictionary, ByVal
dataSourceName As String, ByVal [property] As DataBindingProperties)
```

Parameters

dataSource

An `IDictionary` collection of key-and-value pairs.

dataSourceName

The name of the template data marker at which to insert the values imported from the data source.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindColumnData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$Products
          xlt.BindColumnData(coll,
               "Products",
               xlt.CreateDataBindingProperties());
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$Products
          xlt.BindColumnData(coll, _
               "Products", _
               xlt.CreateDataBindingProperties())
```

ExcelTemplate.BindColumnData(System.Collections.IEnumerable, String, DataBindingProperties)

Description

Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.

Signature: C#

```
public virtual void BindColumnData(System.Collections.IEnumerable dataSource, System.String
dataSourceName, DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindColumnData(ByVal dataSource As System.Collections.IEnumerable, ByVal
dataSourceName As String, ByVal [property] As DataBindingProperties)
```

Parameters

dataSource

An `IEnumerable` collection to use as the data source.

dataSourceName

The name of the template data marker at which to insert the values imported from the data source.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindColumnData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$Products
          xlt.BindColumnData(coll,
               "Products",
               xlt.CreateDataBindingProperties());
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$Products
          xlt.BindColumnData(coll, _
               "Products", _
               xlt.CreateDataBindingProperties())
```

ExcelTemplate.BindData

Overload	Description
BindData(Object()(), String(), String, DataBindingProperties)	Sets a two-dimensional - possibly jagged - array of objects as a template data source.
BindData(Object(,), String(), String, DataBindingProperties)	Sets a rectangular array of objects as a template data source.
BindData(System.Data.IDataReader, String, DataBindingProperties)	Sets an ADO.NET IDataReader as a data source to bind to template data markers.
BindData(System.Data.DataSet, String, DataBindingProperties)	Sets an ADO.NET DataSet as a data source to bind to template data markers.
BindData(System.Data.DataTable, String, DataBindingProperties)	Sets an ADO.NET DataTable as a data source to bind to template data markers.

SharePoint Integration Extension Methods

Overload	Description
BindData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint View as a template data source.
BindData(Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint List as a template data source.

ExcelTemplate.BindData(Object(,), String(), String, DataBindingProperties)

Description

Sets a rectangular array of objects as a template data source.

Signature: C#

```
public void BindData(System.Object[,] arrayData, System.String[] columnNames, System.String
dataSourceName, DataBindingProperties property)
```

Signature: VB.NET

```
Public Sub BindData(ByVal arrayData As Object(,), ByVal columnNames As String(), ByVal dataSourceName
As String, ByVal [property] As DataBindingProperties)
```

Parameters

arrayData

A rectangular array of objects to use as the data source. By default, the first dimension corresponds to row and the second to column (that is,

Object[row,column]).

columnNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `columnNames` is specified, both ordinal field binding and named field binding can be used.

dataSourceName

The name of the set of data markers at which to insert the values imported from the data source. `dataSourceName` must be specified, but can be left as null or an empty string if this is the first data source bound AND the data markers in the template use the short data marker syntax or refer to the datasource by number rather than name. Note: `dataSourceName` does not include a data marker's column name, for example, the `dataSourceName` for `%%=Products.ProductID` is "Products."

property

The DataBindingProperties object which contains information about how the data should be bound to the template.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(Page.MapPath("./ArrayBindingTemplate2.xls"));

          //--- Create an array of names and an array of data
          //--- source values and bind the data source to the
          //--- template data markers
          //--- %%=TwoDimArray.FirstName
          //--- %%=TwoDimArray.LastName
          //--- %%=TwoDimArray.Position
          string[,] twoDimNormal = {
               {"Nancy", "Davolio", "Sales Manager"},
               {"Michael", "Suyama", "HR Representative"},
               {"Adrian", "King", "IS Support"}
               };
          string[] names = {"FirstName", "LastName", "Position"};
          xlt.BindData(twoDimNormal,
               names,
               "TwoDimArray",
               xlt.CreateDataBindingProperties());
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open(Page.MapPath("./ArrayBindingTemplate2.xls"))

          '--- Create an array of names and an array of data
          '--- source values and bind the data source to the
          '--- template data markers
          '--- %%=TwoDimArray.FirstName
          '--- %%=TwoDimArray.LastName
          '--- %%=TwoDimArray.Position
          Dim twoDimNormal(,) As String = New String(,){ _
               {"Nancy", "Davolio", "Sales Manager"}, _
               {"Michael", "Suyama", "HR Representative"}, _
               {"Adrian", "King", "IS Support"}}
          Dim names As String() = {"FirstName", "LastName", "Position"}
          xlt.BindData(twoDimNormal, _
               names, _
               "TwoDimArray", _
               xlt.CreateDataBindingProperties())
```

ExcelTemplate.BindData(Object()(), String(), String, DataBindingProperties)

Description

Sets a two-dimensional - possibly jagged - array of objects as a template data source.

Signature: C#

```
public virtual void BindData(System.Object[][] arrayData, System.String[] columnNames, System.String
dataSourceName, DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindData(ByVal arrayData As Object()(), ByVal columnNames As String(), ByVal
dataSourceName As String, ByVal [property] As DataBindingProperties)
```

Parameters

arrayData

A two-dimensional array of objects to use as the data source. By default, the first dimension corresponds to row and the second to column (that is, Object[row][column]).

columnNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `columnNames` is specified, both ordinal field binding and named field binding can be used.

dataSourceName

The name of the set of data markers at which to insert the values imported from the data source. `dataSourceName` must be specified, but can be left as null or an empty string if this is the first data source bound AND the data markers in the template use the ?short data marker syntax or refer to the datasource by number rather than name. Note: `dataSourceName` does not include a data marker's column name, for example, the `dataSourceName` for `%%=Products.ProductID` is "Products."

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\DataBinding\ArrayBindingTemplate.xls");
          string[][] twodim = {
               new string[]{"Watertown", "MA", "02472"},
               new string[]{"Washington", "DC", "20500"}
               };
          string[] names = {"City", "State", "Zip"};
          xlt.BindData(twodim,
               names,
               "TwoDimArray",
               xlt.CreateDataBindingProperties());
          xlt.PreserveStrings = true;
          xlt.Process();
          xlt.Save(Page.Response,
               "ArrayBinding.xls",
               false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\DataBinding\ArrayBindingTemplate.xls")
          Dim twodim()() As String = New String()() { _
               New String(){"Watertown", "MA", "02472"}, _
               New String(){"Washington", "DC", "20500"}, _
               }
          Dim names As String() = {"City", "State", "Zip"}
          xlt.BindData(twodim, _
               names, _
               "TwoDimArray", _
               xlt.CreateDataBindingProperties())
          xlt.PreserveStrings = True
          xlt.Process()
          xlt.Save(Page.Response, _
              "ArrayBinding.xls", _
               False)
```

ExcelTemplate.BindData(System.Data.DataSet, String, DataBindingProperties)

Description

Sets an ADO.NET DataSet as a data source to bind to template data markers.

Signature: C#

```
public void BindData(System.Data.DataSet data, System.String dataSourceName, DataBindingProperties
property)
```

Signature: VB.NET

```
Public Sub BindData(ByVal data As System.Data.DataSet, ByVal dataSourceName As String, ByVal
[property] As DataBindingProperties)
```

Parameters

source

The DataSet to use as the data source.

dataSourceName

The name of the set of data markers at which to insert the values imported from the data source. `dataSourceName` must be specified, but can be left as null or an empty string if this is the first data source bound AND the data markers in the template use the ?short data marker syntax or refer to the datasource by number rather than name. Note: `dataSourceName` does not include a data marker's column name, for example, the `dataSourceName` for `%%=Products.ProductID` is "Products."

property

The DataBindingProperties object which contains information about how the data should be bound to the template.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();

          OleDbConnection Conn = new OleDbConnection();
          DataSet OrdersDs = null;
          try
          {
               Conn.ConnectionString = Application["connstring"].ToString();

               //--- SQL Query for orders
               string OrdersSQL =
                    "SELECT Orders.OrderID, Customers.CompanyName As Customer, " +
                    "Orders.OrderDate, " +
                    "([Order Details].UnitPrice * [Order Details].Quantity) " +
                    "As [OrderTotal] " +
                    "FROM Orders, [Order Details], Customers " +
                    "WHERE Orders.OrderID=[Order Details].OrderID AND " +
                    "Orders.CustomerID=Customers.CustomerID AND Orders.EmployeeID=?";
               OleDbCommand CmdOrders = new OleDbCommand(OrdersSQL, Conn);
               CmdOrders.Parameters.Add("@EmployeeID", EmployeeId);
               OleDbDataAdapter AdptSales = new OleDbDataAdapter(CmdOrders);
               OrdersDs = new DataSet();
               AdptSales.Fill(OrdersDs, "Orders");
          }
          xlt.BindData(OrdersDs,
               "Orders",
               xlt.CreateDataImportProperties());
```

vb.net

```
Dim xlt As New ExcelTemplate()
          Dim Conn As New OleDbConnection()
          Dim OrdersDs As DataSet = Nothing
          Try
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL Query for orders
               Dim OrdersSQL As String = _
                    "SELECT Orders.OrderID, Customers.CompanyName As Customer, " & _
                    "Orders.OrderDate, " & _
                    "([Order Details].UnitPrice * [Order Details].Quantity) " & _
                    "As [OrderTotal] " & _
                    "FROM Orders, [Order Details], Customers" & _
                    "WHERE Orders.OrderID=[Order Details].OrderID AND " & _
                    "Orders.CustomerID=Customers.CustomerID AND Orders.EmployeeID=?"
               Dim CmdOrders As New OleDbCommand(OrdersSQL, Conn)
               CmdOrders.Parameters.Add("@EmployeeID", EmployeeId)
               Dim AdptSales As New OleDbDataAdapter(CmdOrders)
               OrdersDs = New DataSet()
               AdptSales.Fill(OrdersDs, "Orders")
          End Try
          xlt.BindData(OrdersDs, _
               "Orders", _
               xlt.CreateDataImportProperties())
```

ExcelTemplate.BindData(System.Data.DataTable, String, DataBindingProperties)

Description

Sets an ADO.NET DataTable as a data source to bind to template data markers.

Signature: C#

```
public virtual void BindData(System.Data.DataTable source, System.String dataSourceName,
DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindData(ByVal source As System.Data.DataTable, ByVal dataSourceName As String,
ByVal [property] As DataBindingProperties)
```

Parameters

source

The DataTable to use as the data source.

dataSourceName

The name of the set of data markers at which to insert the values imported from the data source. `dataSourceName` must be specified, but can be left as null or an empty string if this is the first data source bound AND the data markers in the template use the ?short data marker syntax or refer to the datasource by number rather than name. Note: `dataSourceName` does not include a data marker's column name, for example, the `dataSourceName` for `%%=Products.ProductID` is "Products."

property

The DataBindingProperties object which contains information about how the data should be bound to the template.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in

`ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          OleDbConnection Conn = new OleDbConnection();
          DataTable EmployeeDt = null;
          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);
          }
          xlt.BindData(EmployeeDt,
               "Employee",
               xlt.CreateDataBindingProperties());
```

vb.net

```
Dim xlt As New ExcelTemplate()
          Dim Conn As New OleDbConnection()
          Dim EmployeeDt As DataTable = Nothing
          Try
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL Query for employee information
               Dim EmployeeSQL As String = "SELECT FirstName & ' ' & _
                    LastName As Name, Title " & _
                    "FROM Employees WHERE EmployeeID=?"
               Dim CmdEmployee As New OleDbCommand(EmployeeSQL, Conn)
               CmdEmployee.Parameters.Add("@EmployeeID", EmployeeId)
               Dim AdptEmployee As New OleDbDataAdapter(CmdEmployee)
               EmployeeDt = New DataTable()
               AdptEmployee.Fill(EmployeeDt)
          End Try
          xlt.BindData(EmployeeDt, _
               "Employee", _
               xlt.CreateDataBindingProperties())
```

ExcelTemplate.BindData(System.Data.IDataReader, String, DataBindingProperties)

Description

Sets an ADO.NET IDataReader as a data source to bind to template data markers.

Signature: C#

```
public void BindData(System.Data.IDataReader source, System.String dataSourceName,
DataBindingProperties property)
```

Signature: VB.NET

```
Public Sub BindData(ByVal source As System.Data.IDataReader, ByVal dataSourceName As String, ByVal
[property] As DataBindingProperties)
```

Parameters

source

This parameter must be an instance of `System.Data.SqlClient.SqlDataReader` or `System.Data.OleDb.OleDbDataReader` or `Microsoft.AnalysisServices.AdomdClient.AdomdDataReader`, otherwise `BindData` will throw an exception.

dataSourceName

The name of the set of data markers at which to insert the values imported from the data source. `dataSourceName` must be specified, but can be left as null or an empty string if this is the first data source bound AND the data markers in the template use the ?short data marker syntax or refer to the datasource by number rather than name. Note: `dataSourceName` does not include a data marker's column name, for example, the `dataSourceName` for `%%=Products.ProductID` is "Products."

property

The DataBindingProperties object which contains information about how the data should be bound to the template.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
OleDbDataReader CategoryRdr = GetCategoryReader();
          try
          {
               ExcelTemplate xlt = new ExcelTemplate();
               xlt.BindData(CategoryRdr,
                    "Category",
                    ExcelTemplate.CreateDataImportProperties());
          } catch(System.Exception ex) {
          } finally {
               //--- The OleDbDataReader must be open when it's passed
               //--- to ExcelTemplate. Remember to close it after use.
               if(CategoryRdr!=null)
                    CategoryRdr.Close();
          }

          //--- Get an OleDbDataReader containing a list of product categories
          private OleDbDataReader GetCategoryReader()
          {
               OleDbConnection Conn = new OleDbConnection();
               Conn.ConnectionString = Application["connstring"].ToString();

               //--- SQL query for a list of categories
               string CategorySQL = "SELECT CategoryName FROM Categories";
               OleDbCommand Cmd = new OleDbCommand(CategorySQL, Conn);
               Conn.Open();
               return Cmd.ExecuteReader();
          }
```

vb.net

```
Dim CategoryRdr As OleDbDataReader = GetCategoryReader()
          Try
               Dim xlt As New ExcelTemplate()
               xlt.BindData(CategoryRdr, _
                    "Category", _
                    ExcelTemplate.CreateDataBindingProperties())
          Catch ex As System.Exception
          Finally
               '--- The OleDbDataReader must be open when it's passed
               '--- to ExcelTemplate. Remember to close it after use.
               If Not CategoryRdr = Nothing
                    CategoryRdr.Close()
               End If
          End Try

          '--- Get an OleDbDataReader containing a list of product categories
          Private Function GetCategoryReader() As OleDbDataReader
               Dim Conn As New OleDbConnection()
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL query for a list of categories
               Dim CategorySQL As String = "SELECT CategoryName FROM Categories"
               Dim Cmd As New OleDbCommand(CategorySQL, Conn)
               Conn.Open()
               Return Cmd.ExecuteReader()
          End Function
```

ExcelTemplate.BindRowData

Overload	Description
BindRowData(Object(), String(), String, DataBindingProperties)	Sets an array of objects as a data source for a single row in a template.

BindRowData(System.Collections.IDictionary, String, DataBindingProperties)	Sets a row's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.
BindRowData(System.Collections.IEnumerable, String(), String, DataBindingProperties)	Sets a row's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.
BindRowData(System.Data.DataSet, String, DataBindingProperties)	Sets a DataSet as a data source to bind to a row in the template.
BindRowData(System.Data.DataTable, String, DataBindingProperties)	Sets a DataTable as a data source to bind to a row in the template.
BindRowData(System.Data.DataView, String, DataBindingProperties)	Sets a DataView as a data source to bind to a row in the template.

SharePoint Integration Extension Methods

Overload	Description
BindRowData(Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint View as a data source for a single row in a template.
BindRowData(Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint List as a data source for a single row in a template.

ExcelTemplate.BindRowData(Object(), String(), String, DataBindingProperties)

Description

Sets an array of objects as a data source for a single row in a template.

Signature: C#

```
public virtual void BindRowData(System.Object[] rowData, System.String[] columnNames, System.String
dataSourceName, DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindRowData(ByVal rowData As Object(), ByVal columnNames As String(), ByVal
dataSourceName As String, ByVal [property] As DataBindingProperties)
```

Parameters

rowData

An object to use as the data source.

columnNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `columnNames` is specified, both ordinal field binding and named field binding can be used.

dataSourceName

The name of the set of data markers at which to insert the values imported from the data source. `dataSourceName` must be specified, but can be left as null or an empty string if this is the first data source bound AND the data markers in the template use the ?short data marker syntax or refer to the datasource by number rather than name. Note: `dataSourceName` does not include a data marker's column name, for example, the `dataSourceName` for `%%=Products.ProductID` is "Products."

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindRowData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Examples

C#

```
//--- Set a data source for the row of data markers
          //--- %%=Address.Street, %%=Address.City, and
          //--- %%=Address.State
          string[] addressvalues = {"1330 Beacon St.", "Brookline", "MA"};
          string[] addressnames = {"Street", "City", "State"};
          xlt.BindRowData(addressvalues,
               addressnames,
               "Address",
               xlt.CreateDataBindingProperties());
```

vb.net

```
'--- Set a data source for the row of data markers
          '--- %%=Address.Street, %%=Address.City, and
          '--- %%=Address.State
          Dim addressvalues As String() = {"1330 Beacon St.", "Brookline", "MA"}
          Dim addressnames As String() = {"Street", "City", "State"}
          xlt.BindRowData(addressvalues, _
               addressnames, _
               "Address" _
               xlt.CreateDataBindingProperties())
```

ExcelTemplate.BindRowData(System.Collections.IDictionary, String, DataBindingProperties)

Description

Sets a row's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.

Signature: C#

```
public virtual void BindRowData(System.Collections.IDictionary dataSource, System.String
dataSourceName, DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindRowData(ByVal dataSource As System.Collections.IDictionary, ByVal
dataSourceName As String, ByVal [property] As DataBindingProperties)
```

Parameters

dataSource

An `IDictionary` collection of key-and-value pairs to use as the data source.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindRowData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the IDictionary collection to the
          //--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersColl,
               "Orders",
               xlt.CreateDataBindingProperties());
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the IDictionary collection to the
          '--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersColl, _
               "Orders", _
               xlt.CreateDataBindingProperties())
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.BindRowData(System.Collections.IEnumerable, String(), String, DataBindingProperties)

Description

Sets a row's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.

Signature: C#

```
public virtual void BindRowData(System.Collections.IEnumerable dataSource, System.String[]
columnNames, System.String dataSourceName, DataBindingProperties property)
```

Signature: VB.NET

```
Public Overridable Sub BindRowData(ByVal dataSource As System.Collections.IEnumerable, ByVal
columnNames As String(), ByVal dataSourceName As String, ByVal [property] As DataBindingProperties)
```

Parameters

dataSource

An `IEnumerable` collection to use as the data source.

colNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `colNames` is specified, both ordinal field binding and named field binding can be used.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindRowData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the IEnumerable collection to the
          //--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersColl,
               null,
               "Orders",
               xlt.CreateDataBindingProperties());
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate();
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the IEnumerable collection to the
          '--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersColl, _
               null, _
               "Orders", _
               xlt.CreateDataBindingProperties())
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.BindRowData(System.Data.DataSet, String, DataBindingProperties)

Description

Sets a DataSet as a data source to bind to a row in the template.

Signature: C#

```
public void BindRowData(System.Data.DataSet dataSource, System.String dataSourceName,
DataBindingProperties property)
```

Signature: VB.NET

```
Public Sub BindRowData(ByVal dataSource As System.Data.DataSet, ByVal dataSourceName As String, ByVal
[property] As DataBindingProperties)
```

Parameters

source

The DataSet to use as the data source.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindRowData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Only the first row of the DataSet will be used. This method is commonly used to create a set of key-value pairs via a single row of data.

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the DataSet to the
          //--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersDS,
               "Orders",
               xlt.CreateDataBindingProperties());
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the DataSet to the
          '--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersDS, _
               "Orders", _
               xlt.CreateDataBindingProperties())
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.BindRowData(System.Data.DataTable, String, DataBindingProperties)

Description

Sets a DataTable as a data source to bind to a row in the template.

Signature: C#

```
public void BindRowData(System.Data.DataTable dataSource, System.String dataSourceName,
DataBindingProperties property)
```

Signature: VB.NET

```
Public Sub BindRowData(ByVal dataSource As System.Data.DataTable, ByVal dataSourceName As String,
ByVal [property] As DataBindingProperties)
```

Parameters

source

The DataTable to use as the data source.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindRowData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Only the first row of the DataTable will be used. This method is commonly used to create a set of key-value pairs via a single row of data.

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the DataTable to the
          //--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersDT,
               "Orders",
               xlt.CreateDataBindingProperties());
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the DataTable to the
          '--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersDT, _
               "Orders", _
               xlt.CreateDataBindingProperties())
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.BindRowData(System.Data.DataView, String, DataBindingProperties)

Description

Sets a DataView as a data source to bind to a row in the template.

Signature: C#

```
public void BindRowData(System.Data.DataView dataSource, System.String dataSourceName,
DataBindingProperties property)
```

Signature: VB.NET

```
Public Sub BindRowData(ByVal dataSource As System.Data.DataView, ByVal dataSourceName As String, ByVal
[property] As DataBindingProperties)
```

Parameters

source

The DataView to use as the data source.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

property

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindRowData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

Only the first row of the DataView will be used. This method is commonly used to create a set of key-value pairs via a single row of data.

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the DataView to the
          //--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersDV,
               "Orders",
               xlt.CreateDataBindingProperties());
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")
          ...

          '--- Bind the DataView to the
          '--- %%=Orders.[Field] data marker.
          xlt.BindRowData(OrdersDV, _
               "Orders", _
               xlt.CreateDataBindingProperties())
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.ContentType

Description

Sets or returns the generated spreadsheet's MIME content type.

Signature: C#

```
public System.String ContentType{ get; set; }
```

Signature: VB.NET

```
Public Property ContentType() As String
```

Remarks

In most browsers, Microsoft Excel files are mapped to the MIME type "application/vnd.ms-excel" and ContentType is set to this value by default. However, some browsers map Excel files to a different MIME content type, and the browser will not open a file of type "application/vnd.ms-Excel." In this case, set `ContentType` to the appropriate MIME content type.

In Internet Explorer, to find out what MIME content type maps to Excel files:

1. Open the Start menu and select Run.
2. Entire *regedit* to open the Registry Editor.
3. Open HKEY_CLASSES_ROOT\MIME\Database\Content Type. This folder contains a list of MIME content type folders. If you select a folder, you will see the extensions to which the type maps in the right frame.

In Netscape Navigator, to find out what MIME content type maps to Excel files:

1. In the browser window, open the Edit menu and select Preferences...
2. In the Category window select Applications.
3. In the Description window, select an application to display its MIME content type.

`ContentType` is a **Read/Write** property.

Examples

C#

```
xlt.ContentType = "application/unknown";
```

vb.net

```
xlt.ContentType = "application/unknown"
```

ExcelTemplate.CreateDataBindingProperties()

Description

Creates a DataBindingProperties object for assigning the property values associated with one or more data binding calls.

Signature: C#

```
public virtual DataBindingProperties CreateDataBindingProperties()
```

Signature: VB.NET

```
Public Overridable Function CreateDataBindingProperties() As DataBindingProperties
```

Exceptions

ArgumentNullException

`CreateDataBindingProperties` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

A `DataBindingProperties` object is required for each data binding call, including these methods: BindData, BindCellData, BindColumnData, and BindRowData. Hence, CreateDataBindingProperties must be called at least once in any WordTemplate project that binds data to a template.

The same `DataBindingProperties` object can be used for multiple data binding method calls.

Examples

C#

```
//--- Create a DataBindingProperties object to be used
          //--- in a data binding call
          DataBindingProperties dbp = ExcelTemplate.CreateDataBindingProperties();
          dbp.MaxRows = 20;
          xlt.BindData(twodim, names, "TwoDimArray", dbp);
          xlt.Process();
          xlt.Save(Page.Response, "output.xls", false);
```

vb.net

```
'--- Create a DataBindingProperties object to be used
          '--- multiple times in a project
          Dim dbp As DataBindingProperties = ExcelTemplate.CreateDataBindingProperties()
          dbp.MaxRows = 20
          xlt.BindData(twodim, names, "TwoDimArray", dbp)
          xlt.Process()
          xlt.Save(Page.Response, "output.xls", False)
```

ExcelTemplate.DecryptPassword

Description

To decrypt an encrypted template, set DecryptPassword to the password set using EncryptPassword or in Microsoft Excel (Tools -> Options... -> Security).

Signature: C#

```
public System.String DecryptPassword{ get; set; }
```

Signature: VB.NET

```
Public Property DecryptPassword() As String
```

Remarks

Set `DecryptPassword` before calling Open.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.DecryptPassword = "password";
          xlt.Open(Application["templatepath"] +
               @"\DataBinding\EmployeeOrdersTemplate.xls");
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.DecryptPassword = "password"
          xlt.Open(Application["templatepath"] & _
               "\DataBinding\EmployeeOrdersTemplate.xls")
```

ExcelTemplate.EncryptPassword

Description

If you set EncryptPassword, ExcelWriter will encrypt the generated spreadsheet - when Process is called - using RC4 encryption.

Signature: C#

```
public System.String EncryptPassword{ get; set; }
```

Signature: VB.NET

```
Public Property EncryptPassword() As String
```

Remarks

Set `EncryptPassword` to a string no longer than 15 characters. Use this string to decrypt the file, either in Microsoft Excel or by setting DecryptPassword.

Set `EncryptPassword` **before** calling `Process`.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.EncryptPassword = "MyPassword";
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.EncryptPassword = "MyPassword"
```

ExcelTemplate.ExcelTemplate()

Description

Creates a new ExcelTemplate object.

Signature: C#

```
public ExcelTemplate ( )
```

Signature: VB.NET

```
Public Sub New ExcelTemplate ( )
```

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
```

vb.net

```
Dim xlt As New ExcelTemplate()
```

ExcelTemplate.ExcludeMacros

Description

Sets whether ExcelWriter should remove template macros from the generated spreadsheet or include them.

Signature: C#

```
public boolean ExcludeMacros{ get; set; }
```

Signature: VB.NET

```
Public Property ExcludeMacros() As Boolean
```

Remarks

If ExcludeMacros is set to `false`, and the template spreadsheet contains macros, ExcelWriter will preserve the macros in the generated spreadsheet. If `ExcludeMacros` is set to `true`, and the template spreadsheet contains macros, ExcelWriter will not include the macros in the generated spreadsheet. `ExcludeMacros` is set to `false` by default.

`ExcludeMacros` is a **Read/Write** property.

Examples

C#

```
//--- Setting SetExcludeMacros to "true" tells ExcelWriter to
          //--- to remove macros in the template from the generated
          //--- spreadsheet.
          xlt.ExcludeMacros = true;
```

vb.net

```
'--- Setting SetExcludeMacros to "True" tells ExcelWriter to
          '--- to remove macros in the template from the generated
          '--- spreadsheet.
          xlt.ExcludeMacros = True
```

ExcelTemplate.LicenseKey

Description

Sets or returns the 23-character license key that enables ExcelWriter for the current instance.

Signature: C#

```
public System.String LicenseKey{ get; set; }
```

Signature: VB.NET

```
Public Property LicenseKey() As String
```

Remarks

This key does not persist once the code completes execution. If this property is set, ExcelWriter will use this key and ignore any keys in the registry.

Examples

C#

```
string key = xlt.LicenseKey;

          //--- Bind the license key to the data marker
          //--- %%=$ExcelWriterLicenseKey.
          xlt.SetCellDataSource(key, "ExcelWriterLicenseKey");
```

vb.net

```
Dim key As String = xlt.LicenseKey

          '--- Bind the license key to the data marker
          '--- %%=$ExcelWriterLicenseKey
          xlt.SetCellDataSource(key, "ExcelWriterLicenseKey")
```

ExcelTemplate.Open

Overload	Description
Open(ExcelApplication, Workbook)	Passes a spreadsheet from ExcelApplication to ExcelTemplate.
Open(String)	Opens an ExcelWriter template spreadsheet. A template is a file created in Microsoft Excel that contains data markers where data source values will be inserted.

Open(System.IO.Stream)	Opens an ExcelWriter template from a `System.IO.Stream`. An ExcelWriter template is a file created in Microsoft Excel that contains data markers where data source values will be inserted.

SharePoint Extension Methods

Overload	Description
Open(Microsoft.SharePoint.SPDocumentLibrary, String)	Opens a spreadsheet from a SharePoint Document Library.
Open(Microsoft.SharePoint.SPListItem, String)	Opens a spreadsheet from a SharePoint List Item.

ExcelTemplate.Open(ExcelApplication, Workbook)

Description

Passes a spreadsheet from ExcelApplication to ExcelTemplate.

Signature: C#

```
public virtual void Open(ExcelApplication excelApplication, Workbook workbook)
```

Signature: VB.NET

```
Public Overridable Sub Open(ByVal excelApplication As ExcelApplication, ByVal workbook As Workbook)
```

Parameters

excelApplication

The instance of `ExcelApplication` that was used to create the spreadsheet.

workbook

A Workbook object representing the Excel file created with `ExcelApplication`.

Exceptions

ArgumentNullException

Open will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can use `ExcelApplication` to create a document with data markers (a template) and pass the file to `ExcelTemplate` to populate the data markers.

If your template file is the new Office Open XML (.xlsx) format, you cannot open it using `ExcelApplication` in ExcelWriter version 7.

Examples

C#

```
ExcelApplication xla = new ExcelApplication();
          Workbook wb = xla.Create();
          ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(xla, wb);
```

vb.net

```
Dim xla As New ExcelApplication()
          Dim wb As Workbook = xla.Create()
          Dim xlt As New ExcelTemplate()
          xlt.Open(xla, wb)
```

ExcelTemplate.Open(String)

Description

Opens an ExcelWriter template spreadsheet. A template is a file created in Microsoft Excel that contains data markers where data source values will be inserted.

Signature: C#

```
public virtual void Open(System.String fileName)
```

Signature: VB.NET

```
Public Overridable Sub Open(ByVal fileName As String)
```

Parameters

templateFileName

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

Exceptions

ArgumentNullException

Open will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

If your template file is the new Office Open XML (.xlsx) format, you can pass it into the Open method just as you would a BIFF8 (.xls) Excel file. You will need to remember to give the output file a ".xlsx" extension when you call Save, as ExcelWriter cannot convert .xlsx files to .xls.

Examples

C#

```
//--- Open an Excel BIFF8 (.xls) file
          oExcelTemplate.Open(Application["templatepath"] +
               @"\DataBinding\EmployeeOrdersTemplate.xls");

          //--- Open an Office Open XML Excel (.xlsx) file
          oExcelTemplate.Open(Application["templatepath"] +
               @"\DataBinding\EmployeeOrdersTemplate.xlsx");
```

vb.net

```
'--- Open an Excel BIFF8 (.xls) file
          oExcelTemplate.Open(Application("templatepath") & _
               "\DataBinding\EmployeeOrdersTemplate.xls")

          '--- Open an Office Open XML Excel (.xlsx) file
          oExcelTemplate.Open(Application("templatepath") & _
               "\DataBinding\EmployeeOrdersTemplate.xlsx")
```

ExcelTemplate.Open(System.IO.Stream)

Description

Opens an ExcelWriter template from a `System.IO.Stream`. An ExcelWriter template is a file created in Microsoft Excel that contains data markers where data source values will be inserted.

Signature: C#

```
public virtual void Open(System.IO.Stream stream)
```

Signature: VB.NET

```
Public Overridable Sub Open(ByVal stream 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

Open will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

If your template file is the new Office Open XML (.xlsx) format, you can pass it into the ExcelTemplate.Open method just as you would a BIFF8 (.xls) Excel file. You will need to remember to give the output file a ".xlsx" extension when you call Save, as ExcelWriter cannot convert .xlsx files to .xls.

Examples

C#

```
//--- Open an Excel BIFF8 (.xls) file
          FileStream oFileStream = New FileStream(@"C:\Templates\Template.xls", FileMode.Open);
          oExcelTemplate.Open(oFileStream);

          //--- Open an Office Open XML Excel (.xlsx) file
          FileStream oFileStream = New FileStream(@"C:\Templates\Template.xls", FileMode.Open);
          oExcelTemplate.Open(oFileStream);
```

vb.net

```
'--- Open an Excel BIFF8 (.xls) file
          Dim oFileStream As New FileStream("C:\Templates\Template.xls", FileMode.Open)
          oExcelTemplate.Open(oFileStream)

          '--- Open an Office Open XML Excel (.xlsx) file
          Dim oFileStream As New FileStream("C:\Templates\Template.xls", FileMode.Open)
          oExcelTemplate.Open(oFileStream)
```

ExcelTemplate.PreserveStrings

Description

If you set PreserveStrings to `true`, ExcelWriter will preserve numeric strings as strings.

Signature: C#

```
public boolean PreserveStrings{ get; set; }
```

Signature: VB.NET

```
Public Property PreserveStrings() As Boolean
```

Remarks

By default, when populating template data markers, ExcelWriter tries to convert numeric strings to numbers.

`PreserveStrings` is a **Read/Write** property.

This property can be set for individual data markers using the Preserve data marker modifier.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\DataBinding\ArrayBindingTemplate.xls");
          string[][] twodim = {
               new string[]{"Watertown", "MA", "02472"},
               new string[]{"Washington","DC", "20500"}
               };
          string[] names = {"City", "State", "Zip"};
          xlt.BindData(twodim,
               names,
               "TwoDimArray",
               xlt.CreateDataBindingProperties());
          xlt.PreserveStrings = true;
          xlt.Process();
          xlt.Save(Page.Response, "ArrayBinding.xls", false);
```

vb.net

```
Dim xlw As New ExcelTemplate()
          xlt.Open("C:\DataBinding\ArrayBindingTemplate.xls")
          Dim twodim As String()() = { _
               New String(){"Watertown", "MA", "02472"}, _
               New String(){"Washington", "DC", "20500"}, _
               }
          Dim names As String() = {"City", "State", "Zip"}
          xlt.BindData(twodim, _
               names, _
               "TwoDimArray", _
               xlt.CreateDataBindingProperties())
          xlt.PreserveStrings = True
          xlt.Process()
          xlt.Save(Page.Response, "ArrayBinding.xls", False)
```

ExcelTemplate.Process()

Description

The Process method enters data source values in a template's data markers, and creates the output file (the new spreadsheet) in memory. The Save method can then save the output file to disk, stream it to the browser, or both.

Signature: C#

```
public virtual void Process()
```

Signature: VB.NET

```
Public Overridable Sub Process()
```

Remarks

Call `Process` after SetDataSource and before Save.

You can call `Process` only once for each instance of ExcelTemplate.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(Application["templatepath"] +
               @"\DataBinding\ArrayBindingTemplate2.xls");

          //--- Set a data source for the data marker "SimpleArray."
          string[] onedim = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"};
          xlt.SetColumnDataSource(onedim, "SimpleArray");

          //--- Process the template to populate it with the data source data.
          xlt.Process();

          //--- Stream the processed document to the client.
          xlt.Save(Page.Response, "ArrayBinding.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open(Application("templatepath") & _
               "\DataBinding\ArrayBindingTemplate2.xls")

          '--- Set a data source for the data marker "SimpleArray."
          Dim onedim As String() = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"}
          xlt.SetColumnDataSource(onedim, "SimpleArray")

          '--- Process the template to populate it with the data source data.
          xlt.Process()

          '--- Stream the processed document to the client.
          xlt.Save(Page.Response, "ArrayBinding.xls", False)
```

ExcelTemplate.RemoveExtraDataMarkers

Description

Sets whether ExcelWriter should remove data markers in the template that do not bind to data sources in code.

Signature: C#

```
public boolean RemoveExtraDataMarkers{ get; set; }
```

Signature: VB.NET

```
Public Property RemoveExtraDataMarkers() As Boolean
```

Remarks

By default, if a data marker does not bind to a data source, ExcelWriter will throw an exception. If `RemoveExtraDataMarkers` is set to `true`, ExcelWriter will remove extra data markers silently.

If a data marker includes the "Optional" modifier, and does not bind to a data source, ExcelWriter will remove it without error even if `RemoveExtraDataMarkers` is set to `false`.

`RemoveExtraDataMarkers` is a **Read/Write** property.

Examples

C#

```
//--- Setting RemoveDataMarkers to 'true' tells
          //--- ExcelWriter to remove data markers that do not
          //--- bind to a data source.
          xlt.RemoveExtraDataMarkers = true;
```

vb.net

```
'--- Setting RemoveDataMarkers to 'True' tells
          '--- ExcelWriter to remove data markers that do not
          '--- bind to a data source.
          xlt.RemoveExtraDataMarkers = True
```

ExcelTemplate.Save

Overload	Description
Save(String)	Saves the generated Excel file on the server. The ASPNET account (IIS5), NETWORK SERVICE account (IIS6 or IIS7), or the authenticated user must have Write access to the destination directory specified by the `outputFileName` parameter.
Save(System.IO.Stream)	Generates an Excel binary file and streams it to the specified `System.IO.Stream` or, a class derived from `System.IO.Stream` (for example, `System.IO.FileStream`). If you pass Save a `System.IO.FileStream`, ExcelWriter will save the generated file on the server. If you pass `Save Response.OutputStream`, ExcelWriter will stream the the generated file to the client.
Save(System.Web.HttpResponse)	If you pass Save an `HttpResponse` object object, ExcelWriter will stream the generated file to the client.
Save(System.Web.HttpResponse, String, Boolean)	If you pass Save an HttpResponse object, ExcelWriter 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 Excel.

SharePoint Integration Extension Methods

Overload	Description
Save(Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Generates an Excel binary or OOXML file and saves it to a SharePoint Document Library.
Save(Microsoft.SharePoint.SPListItem, String)	Generates an Excel binary or OOXML file and saves it as an attachment to a SharePoint list item.

ExcelTemplate.Save(String)

Description

Saves the generated Excel file on the server. The ASPNET account (IIS5), NETWORK SERVICE account (IIS6 or IIS7), or the authenticated user must have Write access to the destination directory specified by the `outputFileName` parameter.

Signature: C#

```
public virtual void Save(System.String fileName)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal fileName As String)
```

Parameters

outputFileName

Specifies a complete path and file name for the generated file. ExcelWriter will save the file to this location. If a file with the same name exists, it will be overwritten by the new Excel file.

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

ExcelWriter allows you to save in both the Excel 97-03 BIFF8 format (.xls) or the new Office Open XML (.xlsx) format. The template file must be of the expected output format. Hence, if you wish to output .xls files, you must start with a .xls template, and if you wish to output .xlsx files, you must start with a .xlsx template. ExcelWriter does not support the creation or editing of .xlsx files with the ExcelApplication object.

You can call Save more than once for a single instance of ExcelTemplate. 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#

```
//--- Save a .xls file
          xlTemplate.Save(@"C:\Reports\Orders.xls");

          //--- Save a .xlsx file
          xlTemplate.Save(@"C:\Reports\Orders.xlsx");
```

vb.net

```
'--- Save a .xls file
          xlTemplate.Save("c:\Reports\Orders.xls")

          '--- Save a .xlsx file
          xlTemplate.Save("c:\Reports\Orders.xlsx")
```

ExcelTemplate.Save(System.IO.Stream)

Description

Generates an Excel binary file and streams it to the specified `System.IO.Stream` or, a class derived from `System.IO.Stream` (for example, `System.IO.FileStream`). If you pass Save a `System.IO.FileStream`, ExcelWriter will save the generated file on the server. If you pass `Save` `Response.OutputStream`, ExcelWriter will stream the the generated file to the client.

Signature: C#

```
public virtual void Save(System.IO.Stream stream)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal stream As System.IO.Stream)
```

Parameters

outputStream

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.

ArgumentException

Remarks

ExcelWriter allows you to save in both the Excel 97-03 BIFF8 format (.xls) or the new Office Open XML (.xlsx) format. The template file must be of the expected output format. Hence, if you wish to output .xls files, you must start with a .xls template, and if you wish to output .xlsx files, you must start with a .xlsx template. ExcelWriter does not support the creation or editing of .xlsx files with the ExcelApplication object.

You can call `Save` more than once for a single instance of ExcelTemplate. 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#

```
//--- FileStream and FileMode are in the System.IO namespace
          //--- This stream is for a .xls template
          FileStream fstream = new FileStream(@"C:\temp\outfile.xls", FileMode.Create);

          //--- Pass the FileStream to ExcelTemplate
          xlTemplate.Save(fstream);

          //--- Close the FileStream (could be in a finally block)
          fstream.Close();

          //--- FileStream and FileMode are in the System.IO namespace
          //--- This stream is for a .xlsx template
          FileStream fstream = new FileStream(@"C:\temp\outfile.xlsx", FileMode.Create);

          //--- Pass the FileStream to ExcelTemplate
          xlTemplate.Save(fstream);

          //--- Close the FileStream (could be in a finally block)
          fstream.Close();
```

vb.net

```
'--- FileStream and FileMode are in the System.IO namespace
          '--- This stream is for a .xls template
          Dim fstream As New FileStream("C:\temp\outfile.xls", FileMode.Create)

          '--- Pass the FileStream to ExcelTemplate
          xlTemplate.Save(fstream)

          '--- Close the FileStream (could be in a finally block)
          fstream.Close()

          '--- FileStream and FileMode are in the System.IO namespace
          '--- This stream is for a .xlsx template
          Dim fstream As New FileStream("C:\temp\outfile.xlsx", FileMode.Create)

          '--- Pass the FileStream to ExcelTemplate
          xlTemplate.Save(fstream)

          '--- Close the FileStream (could be in a finally block)
          fstream.Close()
```

ExcelTemplate.Save(System.Web.HttpResponse, String, Boolean)

Description

If you pass Save an HttpResponse object, ExcelWriter 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 Excel.

Signature: C#

```
public virtual void Save(System.Web.HttpResponse response, System.String attachmentName, boolean
openInBrowser)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal response As System.Web.HttpResponse, ByVal attachmentName As String,
ByVal openInBrowser As Boolean)
```

Parameters

response

A `System.Http.Response` object, usually `Page.Response`

attachmentName

Specifies a name for the generated Excel 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 Excel. By default, the file will open in the browser window.

> ⚠ not all browsers can embed an Excel file in the browser window

Remarks

ExcelWriter allows you to save in both the Excel 97-03 BIFF8 format (.xls) or the new Office Open XML (.xlsx) format. The template file must be of the expected output format. Hence, if you wish to output .xls files, you must start with a .xls template, and if you wish to output .xlsx files, you must start with a .xlsx template. ExcelWriter does not support the creation or editing of .xlsx files with the ExcelApplication object.

You can call `Save` more than once for a single instance of ExcelTemplate. 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 Excel

When you pass an HttpResponse object to `Save`, ExcelWriter will stream the generated Excel 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 Excel file, the file will open in Microsoft Excel.

C#

```
//--- Stream to the client
            //--- "Output.xls" will appear in the Save As dialog
            //--- The file will open in Excel
            xlTemplate.Save(Page.Response, "Output.xls", false);

            //--- Stream to the client
            //--- "Output.xlsx" will appear in the Save As dialog
            //--- The file will open in Excel 2007 only
            xlTemplate.Save(Page.Response, "Output.xlsx", false);
```

vb.net

```
'--- Stream to the client
            '--- "Output.xls" will appear in the Save As dialog
            '--- The file will open in Excel
            xlTemplate.Save(Page.Response, "Output.xls", False)

            '--- Stream to the client
            '--- "Output.xlsx" will appear in the Save As dialog
            '--- The file will open in Excel 2007 only
            xlTemplate.Save(Page.Response, "Output.xlsx", False)
```

Stream to the client and open in the browser window

When you pass an HttpResponse object to `Save`, ExcelWriter will stream the generated Excel 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 Excel file, the file will open in the browser window.

C#

```
//--- Stream to the client
            //--- "Output.xls" will appear in the Save As dialog
            //--- The file will open in the browser's Excel plug-in
            xlTemplate.Save(Page.Response, "Output.xls", true);
            ...

            //--- Stream to the client
            //--- "Output.xlsx" will appear in the Save As dialog
            //--- The file will open in the browser's Excel 2007 plug-in
            xlTemplate.Save(Page.Response, "Output.xlsx", true);
```

vb.net

```
'--- Stream to the client
            '--- "Output.xls" will appear in the Save As dialog
            '--- The file will open in the browser's Excel plug-in
            xlTemplate.Save(Page.Response, "Output.xls", True)
            ...

            '--- Stream to the client
            '--- "Output.xlsx" will appear in the Save As dialog
            '--- The file will open in the browser's Excel 2007 plug-in
            xlTemplate.Save(Page.Response, "Output.xlsx", True)
```

ExcelTemplate.Save(System.Web.HttpResponse)

Description

If you pass Save an `HttpResponse` object object, ExcelWriter will stream the generated file to the client.

Signature: C#

```
public virtual void Save(System.Web.HttpResponse response)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal response As System.Web.HttpResponse)
```

Parameters

response

A `System.Http.Response` object, usually `Page.Response`.

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

If the user chooses to open (rather than save) the file, it will open in the browser window. ExcelWriter will set a default name for the file. To set a different file name and/or to open the file in Microsoft Excel, use the signature Save(HttpResponse,string,bool).

You can call `Save` more than once for a single instance of ExcelTemplate. 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#

```
xlTemplate.Save(Page.Response);
```

vb.net

```
xlTemplate.Save(Page.Response)
```

ExcelTemplate.SetCellDataSource(Object, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindCellData(Object, String, DataBindingProperties) method.

Description

Sets an object as a data source for a single cell in a template.

Signature: C#

```
[Obsolete("SetCellDataSource is deprecated, use BindCellData instead")]
 public virtual void SetCellDataSource(System.Object cellData, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetCellDataSource is deprecated, use BindCellData instead")> _
 Public Overridable Sub SetCellDataSource(ByVal cellData As Object, ByVal dataSourceName As String)
```

Parameters

cellData

An object to use as the data source.

dataMarkerName

The name of the template data marker at which to insert the data source value. This parameter must be specified, but can be null or an empty string when setting the first data source to bind to the template.

Exceptions

ArgumentNullException

`SetCellDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
//--- Set a cell data source for the data marker
          //--- %%=$Company
          xlt.SetCellDataSource("SoftArtisans", "Company");
```

vb.net

```
'--- Set a cell data source for the data marker
          '--- %%=$Company
          xlt.SetCellDataSource("SoftArtisans", "Company")
```

ExcelTemplate.SetColumnDataSource

Overload	Description
SetColumnDataSource(Object(), String)	Sets an array of objects as a data source for a single column in the template. You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, ExcelTemplate.SetRowDataSource, and ExcelTemplate.SetDataSource.
SetColumnDataSource(Object(), String, Int32)	Sets an array of objects as a data source for a single column in the template. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.
SetColumnDataSource(System.Collections.IDictionary, String)	Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.
SetColumnDataSource(System.Collections.IDictionary, String, Int32)	Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.
SetColumnDataSource(System.Collections.IEnumerable, String)	Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.
SetColumnDataSource(System.Collections.IEnumerable, String, Int32)	Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.

ExcelTemplate.SetColumnDataSource(Object(), String, Int32)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindColumnData(Object(), String, DataBindingProperties) method.

Description

Sets an array of objects as a data source for a single column in the template. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.

Signature: C#

```
[Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")]
 public virtual void SetColumnDataSource(System.Object[] columnData, System.String dataSourceName, int
maxRows)
```

Signature: VB.NET

```
<Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")> _
 Public Overridable Sub SetColumnDataSource(ByVal columnData As Object(), ByVal dataSourceName As
String, ByVal maxRows As Integer)
```

Parameters

columnData

An array of objects to use as the data source.

dataSourceName

The name of the template data marker at which to insert the values imported from 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 ExcelTemplate.ALL_ROWS .

Exceptions

ArgumentNullException

`SetColumnDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, ExcelTemplate.SetRowDataSource, and SetDataSource.

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$SimpleArray
          //--- Specify that only 2 rows should be imported
          string[] onedim = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"};
          xlt.SetColumnDataSource(onedim, "SimpleArray", 2);
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$SimpleArray
          '--- Specify that only 2 rows should be imported
          Dim onedim As String() = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"}
          xlt.SetColumnDataSource(onedim, "SimpleArray", 2)
```

ExcelTemplate.SetColumnDataSource(Object(), String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindColumnData(Object(), String, DataBindingProperties) method.

Description

Sets an array of objects as a data source for a single column in the template. You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, ExcelTemplate.SetRowDataSource, and ExcelTemplate.SetDataSource.

Signature: C#

```
[Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")]
 public virtual void SetColumnDataSource(System.Object[] columnData, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")> _
 Public Overridable Sub SetColumnDataSource(ByVal columnData As Object(), ByVal dataSourceName As
String)
```

Parameters

columnData

An array of objects to use as the data source.

dataSourceName

The name of the template data marker at which to insert the values imported from the data source.

Exceptions

ArgumentNullException

`SetColumnDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$SimpleArray
          string[] onedim = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"};
          xlt.SetColumnDataSource(onedim, "SimpleArray");
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$SimpleArray
          Dim onedim As String() = {"SoftArtisans", "OfficeWriter", "ExcelTemplate"}
          xlt.SetColumnDataSource(onedim, "SimpleArray")
```

ExcelTemplate.SetColumnDataSource(System.Collections.IDictionary, String, Int32)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindColumnData(System.Collections.IDictionary, String, DataBindingProperties) method.

Description

Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.

Signature: C#

```
[Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")]
 public virtual void SetColumnDataSource(System.Collections.IDictionary dataSource, System.String
dataSourceName, int maxRows)
```

Signature: VB.NET

```
<Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")> _
 Public Overridable Sub SetColumnDataSource(ByVal dataSource As System.Collections.IDictionary, ByVal
dataSourceName As String, ByVal maxRows As Integer)
```

Parameters

dataSource

An `IDictionary` collection of key-and-value pairs.

dataSourceName

The name of the template data marker at which to insert the values imported from 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 ExcelTemplate.ALL_ROWS .

Exceptions

ArgumentNullException

`SetColumnDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, ExcelTemplate.SetRowDataSource, and SetDataSource.

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$Products
          xlt.SetColumnDataSource(coll, "Products", 7);
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$Products
          xlt.SetColumnDataSource(coll, "Products", 7)
```

ExcelTemplate.SetColumnDataSource(System.Collections.IDictionary, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindColumnData(System.Collections.IDictionary, String, DataBindingProperties) method.

Description

Sets a column's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.

Signature: C#

```
[Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")]
 public virtual void SetColumnDataSource(System.Collections.IDictionary dataSource, System.String
dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")> _
 Public Overridable Sub SetColumnDataSource(ByVal dataSource As System.Collections.IDictionary, ByVal
dataSourceName As String)
```

Parameters

dataSource

An `IDictionary` collection of key-and-value pairs.

dataSourceName

The name of the template data marker at which to insert the values imported from the data source.

Exceptions

ArgumentNullException

`SetColumnDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$Products
          xlt.SetColumnDataSource(coll, "Products");
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$Products
          xlt.SetColumnDataSource(coll, "Products")
```

ExcelTemplate.SetColumnDataSource(System.Collections.IEnumerable, String, Int32)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindColumnData(System.Collections.IEnumerable, String, DataBindingProperties) method.

Description

Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection. The maximum number of rows to return from the data source is defined by the `MaxRows` parameter.

Signature: C#

```
[Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")]
 public virtual void SetColumnDataSource(System.Collections.IEnumerable columnData, System.String
dataSourceName, int maxRows)
```

Signature: VB.NET

```
<Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")> _
 Public Overridable Sub SetColumnDataSource(ByVal columnData As System.Collections.IEnumerable, ByVal
dataSourceName As String, ByVal maxRows As Integer)
```

Parameters

dataSource

An `IEnumerable` collection to use as the data source.

dataSourceName

The name of the template data marker at which to insert the values imported from 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 ExcelTemplate.ALL_ROWS .

Exceptions

ArgumentNullException

`SetColumnDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, ExcelTemplate.SetRowDataSource, and ExcelTemplate.SetDataSource.

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$Products
          xlt.SetColumnDataSource(coll, "Products", 7);
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$Products
          xlt.SetColumnDataSource(coll, "Products", 7)
```

ExcelTemplate.SetColumnDataSource(System.Collections.IEnumerable, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindColumnData(System.Collections.IEnumerable, String, DataBindingProperties) method.

Description

Sets a column's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.

Signature: C#

```
[Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")]
 public virtual void SetColumnDataSource(System.Collections.IEnumerable columnData, System.String
dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetColumnDataSource is deprecated, use BindColumnData instead")> _
 Public Overridable Sub SetColumnDataSource(ByVal columnData As System.Collections.IEnumerable, ByVal
dataSourceName As String)
```

Parameters

dataSource

An `IEnumerable` collection to use as the data source.

dataSourceName

The name of the template data marker at which to insert the values imported from the data source.

Exceptions

ArgumentNullException

`SetColumnDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
//--- Set a column data source for the data marker
          //--- %%=$Products
          xlt.SetColumnDataSource(coll, "Products");
```

vb.net

```
'--- Set a column data source for the data marker
          '--- %%=$Products
          xlt.SetColumnDataSource(coll, "Products")
```

ExcelTemplate.SetDataSource

Overload	Description
SetDataSource(Object()(), String(), String)	Sets a two-dimensional - possibly jagged - array of objects as a template data source.
SetDataSource(Object()(), String(), String, Int32, Boolean)	**(Deprecated)** Sets a two-dimensional - possibly jagged - array of objects as a template data source.
SetDataSource(Object(,), String(), String)	Sets a rectangular array of objects as a template data source.
SetDataSource(Object(,), String(), String, Int32, Boolean)	Sets a rectangular array of objects as a template data source.
SetDataSource(System.Data.IDataReader, String)	Sets an ADO.NET IDataReader as a data source to bind to template data markers.
SetDataSource(System.Data.IDataReader, String, Int32)	Sets an IDataReader as a data source to bind to template data markers.
SetDataSource(System.Data.DataSet, String)	Sets an ADO.NET DataSet as a data source to bind to template data markers.
SetDataSource(System.Data.DataSet, String, Int32)	Sets an ADO.NET DataSet as a data source to bind to template data markers.
SetDataSource(System.Data.DataTable, String)	Sets an ADO.NET DataTable as a data source to bind to template data markers.
SetDataSource(System.Data.DataTable, String, Int32)	Sets an ADO.NET DataTable as a data source to bind to template data markers.

ExcelTemplate.SetDataSource(Object(,), String(), String, Int32, Boolean)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(Object(,), String(), String, DataBindingProperties) method.

Description

Sets a rectangular array of objects as a template data source.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public void SetDataSource(System.Object[,] arrayData, System.String[] columnNames, System.String
dataSourceName, int maxRows, boolean transposeToRowCol)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Sub SetDataSource(ByVal arrayData As Object(,), ByVal columnNames As String(), ByVal
dataSourceName As String, ByVal maxRows As Integer, ByVal transposeToRowCol As Boolean)
```

Parameters

arrayData

A rectangular array of objects to use as the data source. By default, the first dimension corresponds to row and the second to column (that is, Object[row,column]).

columnNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `columnNames` is specified, both ordinal field binding and named field binding can be used.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant ExcelTemplate.ALL_ROWS.

transpose

If `transpose` is set to `true`, the array is treated as Object[column][row]. By default, `transpose` is set to `false`.

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(Page.MapPath("./ArrayBindingTemplate2.xls"));

          //--- Create an array of names and an array of data
          //--- source values and bind the data source to the
          //--- template data markers
          //--- %%=TwoDimArray.FirstName
          //--- %%=TwoDimArray.LastName
          //--- %%=TwoDimArray.Position
          string[,] twoDimTranspose = {
               {"Nancy", "Michael", "Adrian"},
               {"Davolio", "Suyama", "King"},
               {"Sales Manager", "HR Representative", "IS Support"}
               };
          string[] names = {"FirstName", "LastName", "Position"};
          xlt.SetDataSource(twoDimTranspose,
               names,
               "TwoDimTrans",
               ExcelTemplate.ALL_ROWS,
               true);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open(Page.MapPath("ArrayBindingTemplate2.xls"));

          '--- Create an array of names and an array of data
          '--- source values and bind the data source to the
          '--- template data markers
          '--- %%=TwoDimArray.FirstName
          '--- %%=TwoDimArray.LastName
          '--- %%=TwoDimArray.Position
          Dim twoDimTranspose(,) As String = { _
               {"Nancy", "Michael", "Adrian"}, _
               {"Davolio", "Suyama", "King"}, _
               {"Sales Manager", "HR Representative", "IS Support"} _
               }
          names As String() = {"FirstName", "LastName", "Position"}
          xlt.SetDataSource(twoDimTranspose, _
               names, _
               "TwoDimTrans", _
               ExcelTemplate.ALL_ROWS, _
               True)
```

ExcelTemplate.SetDataSource(Object(,), String(), String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(Object(,), String(), String, DataBindingProperties) method.

Description

Sets a rectangular array of objects as a template data source.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public void SetDataSource(System.Object[,] arrayData, System.String[] columnNames, System.String
dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Sub SetDataSource(ByVal arrayData As Object(,), ByVal columnNames As String(), ByVal
dataSourceName As String)
```

Parameters

arrayData

A rectangular array of objects to use as the data source. By default, the first dimension corresponds to row and the second to column (that is, Object[row,column]).

columnNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `columnNames` is specified, both ordinal field binding and named field binding can be used.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(Page.MapPath("./ArrayBindingTemplate2.xls"));

          //--- Create an array of names and an array of data
          //--- source values and bind the data source to the
          //--- template data markers
          //--- %%=TwoDimArray.FirstName
          //--- %%=TwoDimArray.LastName
          //--- %%=TwoDimArray.Position
          string[,] twoDimNormal = {
               {"Nancy", "Davolio", "Sales Manager"},
               {"Michael", "Suyama", "HR Representative"},
               {"Adrian", "King", "IS Support"}
               };
          string[] names = {"FirstName", "LastName", "Position"};
          xlt.SetDataSource(twoDimNormal, names, "TwoDimArray");
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open(Page.MapPath("./ArrayBindingTemplate2.xls"))

          '--- Create an array of names and an array of data
          '--- source values and bind the data source to the
          '--- template data markers
          '--- %%=TwoDimArray.FirstName
          '--- %%=TwoDimArray.LastName
          '--- %%=TwoDimArray.Position
          Dim twoDimNormal(,) As String = New String(,){ _
               {"Nancy", "Davolio", "Sales Manager"}, _
               {"Michael", "Suyama", "HR Representative"}, _
               {"Adrian", "King", "IS Support"} _
               }
          Dim names As String() = {"FirstName", "LastName", "Position"}
          xlt.SetDataSource(twoDimNormal, names, "TwoDimArray")
```

ExcelTemplate.SetDataSource(Object()(), String(), String, Int32, Boolean)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(Object()(), String(), String, DataBindingProperties) method.

Description

(Deprecated) Sets a two-dimensional - possibly jagged - array of objects as a template data source.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public virtual void SetDataSource(System.Object[][] arrayData, System.String[] columnNames,
System.String dataSourceName, int maxRows, boolean transposeToRowCol)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Overridable Sub SetDataSource(ByVal arrayData As Object()(), ByVal columnNames As String(),
ByVal dataSourceName As String, ByVal maxRows As Integer, ByVal transposeToRowCol As Boolean)
```

Parameters

arrayData

A two-dimensional array of objects to use as the data source. By default, the first dimension corresponds to row and the second to column (that is, Object[row][column]).

columnNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `columnNames` is specified, both ordinal field binding and named field binding can be used.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant ExcelTemplate.ALL_ROWS.

transpose

If `transpose` is set to `true`, the array is treated as Object[column][row]. By default, `transpose` is set to `false`.

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\DataBinding\ArrayBindingTemplate.xls");
          string[][] twodim = {
               new string[]{"Watertown", "MA", "02472"},
               new string[]{"Washington", "DC", "20500"}
               };
          string[] names = {"City", "State", "Zip"};
          xlt.SetDataSource(twodim, names, "TwoDimArray", ExcelTemplate.ALL_ROWS, false);
          xlt.PreserveStrings = true; xlt.Process();
          xlt.Save(Page.Response, "ArrayBinding.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\DataBinding\ArrayBindingTemplate.xls")
          Dim twodim()() As String = New String()(){ _
               New String(){"Watertown", "MA", "02472"}, _
               New String(){"Washington", "DC", "20500"} _
               }
          Dim names As String() = {"City", "State", "Zip"}
          xlt.SetDataSource(twodim, names, "TwoDimArray", ExcelTemplate.ALL_ROWS, False)
          xlt.PreserveStrings = True
          xlt.Process()
          xlt.Save(Page.Response, "ArrayBinding.xls", False)
```

ExcelTemplate.SetDataSource(Object()(), String(), String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(Object()(), String(), String, DataBindingProperties) method.

Description

Sets a two-dimensional - possibly jagged - array of objects as a template data source.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public virtual void SetDataSource(System.Object[][] arrayData, System.String[] columnNames,
System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Overridable Sub SetDataSource(ByVal arrayData As Object()(), ByVal columnNames As String(),
ByVal dataSourceName As String)
```

Parameters

arrayData

A two-dimensional array of objects to use as the data source. By default, the first dimension corresponds to row and the second to column (that is, Object[row][column]).

columnNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `columnNames` is specified, both ordinal field binding and named field binding can be used.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\DataBinding\ArrayBindingTemplate.xls");
          string[][] twodim = {
               new string[]{"Watertown", "MA", "02472"},
               new string[]{"Washington", "DC", "20500"}
               };
          string[] names = {"City", "State", "Zip"};
          xlt.SetDataSource(twodim, names, "TwoDimArray");
          xlt.PreserveStrings = true;
          xlt.Process();
          xlt.Save(Page.Response, "ArrayBinding.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\DataBinding\ArrayBindingTemplate.xls")
          Dim twodim()() As String = New String()() { _
               New String(){"Watertown","MA","02472"}, _
               New String(){"Washington","DC","20500"} _
               }
          Dim names As String() = {"City", "State", "Zip"}
          xlt.SetDataSource(twodim, names, "TwoDimArray")
          xlt.PreserveStrings = True
          xlt.Process()
          xlt.Save(Page.Response, "ArrayBinding.xls", False)
```

ExcelTemplate.SetDataSource(System.Data.DataSet, String, Int32)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(System.Data.DataSet, String, DataBindingProperties) method.

Description

Sets an ADO.NET DataSet as a data source to bind to template data markers.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public void SetDataSource(System.Data.DataSet data, System.String dataSourceName, int maxRows)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Sub SetDataSource(ByVal data As System.Data.DataSet, ByVal dataSourceName As String, ByVal
maxRows As Integer)
```

Parameters

source

The DataSet to use as the data source.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant ExcelTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          OleDbConnection Conn = new OleDbConnection();
          DataSet OrdersDs = null;
          try
          {
               Conn.ConnectionString = Application["connstring"].ToString();

               //--- SQL Query for orders
               string OrdersSQL = "SELECT Orders.OrderID, Customers.CompanyName " +
                    "As Customer, " +
                    "Orders.OrderDate, " +
                    "([Order Details].UnitPrice * [Order Details].Quantity) As [OrderTotal] " +
                    "FROM Orders, [Order Details], Customers " +
                    "WHERE Orders.OrderID=[Order Details].OrderID AND " +
                    "Orders.CustomerID=Customers.CustomerID AND Orders.EmployeeID=?";
               OleDbCommand CmdOrders = new OleDbCommand(OrdersSQL, Conn);
               CmdOrders.Parameters.Add("@EmployeeID", EmployeeId);
               OleDbDataAdapter AdptSales = new OleDbDataAdapter(CmdOrders);
               OrdersDs = new DataSet();
               AdptSales.Fill(OrdersDs, "Orders", ExcelTemplate.ALL_ROWS);
          }
          xlt.SetDataSource(OrdersDs, "Orders");
```

vb.net

```
Dim xlt As New ExcelTemplate()
          Dim Conn As New OleDbConnection()
          Dim OrdersDs As DataSet = Nothing
          Try
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL Query for orders
               Dim OrdersSQL As String = "SELECT Orders.OrderID, Customers.CompanyName " & _
                    "As Customer, " & _
                    "Orders.OrderDate, " & _
                    "([Order Details].UnitPrice * [Order Details].Quantity) As [OrderTotal] " & _
                    "FROM Orders, [Order Details], Customers " & _
                    "WHERE Orders.OrderID=[Order Details].OrderID AND " & _
                    "Orders.CustomerID=Customers.CustomerID AND Orders.EmployeeID=?"
               Dim CmdOrders As New OleDbCommand(OrdersSQL, Conn)
               CmdOrders.Parameters.Add("@EmployeeID", EmployeeId)
               Dim AdptSales As New OleDbDataAdapter(CmdOrders)
               OrdersDs = New DataSet()
               AdptSales.Fill(OrdersDs, "Orders", ExcelTemplate.ALL_ROWS)
          End Try
          xlt.SetDataSource(OrdersDs, "Orders")
```

ExcelTemplate.SetDataSource(System.Data.DataSet, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(System.Data.DataSet, String, DataBindingProperties) method.

Description

Sets an ADO.NET DataSet as a data source to bind to template data markers.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public void SetDataSource(System.Data.DataSet data, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Sub SetDataSource(ByVal data As System.Data.DataSet, ByVal dataSourceName As String)
```

Parameters

source

The DataSet to use as the data source.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          OleDbConnection Conn = new OleDbConnection();
          DataSet OrdersDs = null;
          try
          {
               Conn.ConnectionString = Application["connstring"].ToString();

               //--- SQL Query for orders string
               OrdersSQL = "SELECT Orders.OrderID, Customers.CompanyName " +
                    "As Customer, " +
                    "Orders.OrderDate, " +
                    "([Order Details].UnitPrice * [Order Details].Quantity) " +
                    "As [OrderTotal] " +
                    "FROM Orders, [Order Details], Customers " +
                    "WHERE Orders.OrderID=[Order Details].OrderID AND " +
                    "Orders.CustomerID=Customers.CustomerID AND Orders.EmployeeID=?";
               OleDbCommand CmdOrders = new OleDbCommand(OrdersSQL, Conn);
               CmdOrders.Parameters.Add("@EmployeeID", EmployeeId);
               OleDbDataAdapter AdptSales = new OleDbDataAdapter(CmdOrders);
               OrdersDs = new DataSet();
               AdptSales.Fill(OrdersDs, "Orders");
          }
          xlt.SetDataSource(OrdersDs, "Orders");
```

vb.net

```
Dim xlt As New ExcelTemplate()
          Dim Conn As New OleDbConnection()
          Dim OrdersDs As DataSet = Nothing
          Try
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL Query for orders
               Dim OrdersSQL As String = "SELECT Orders.OrderID, Customers.CompanyName " & _
                    "As Customer, " & _
                    "Orders.OrderDate, " & _
                    "([Order Details].UnitPrice * [Order Details].Quantity) " & _
                    "As [OrderTotal] " & _
                    "FROM Orders, [Order Details], Customers " & _
                    "WHERE Orders.OrderID=[Order Details].OrderID AND " & _
                    "Orders.CustomerID=Customers.CustomerID AND Orders.EmployeeID=?"
               Dim CmdOrders As New OleDbCommand(OrdersSQL, Conn)
               CmdOrders.Parameters.Add("@EmployeeID", EmployeeId)
               Dim AdptSales As New OleDbDataAdapter(CmdOrders)
               OrdersDs = New DataSet()
               AdptSales.Fill(OrdersDs, "Orders")
          End Try
          xlt.SetDataSource(OrdersDs, "Orders")
```

ExcelTemplate.SetDataSource(System.Data.DataTable, String, Int32)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(System.Data.DataTable, String, DataBindingProperties) method.

Description

Sets an ADO.NET DataTable as a data source to bind to template data markers.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public virtual void SetDataSource(System.Data.DataTable source, System.String dataSourceName, int
maxRows)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Overridable Sub SetDataSource(ByVal source As System.Data.DataTable, ByVal dataSourceName As
String, ByVal maxRows As Integer)
```

Parameters

source

The DataTable to use as the data source.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant ExcelTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          OleDbConnection Conn = new OleDbConnection();
          DataTable EmployeeDt = null;
          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);
          }
          xlt.SetDataSource(EmployeeDt,
               "Employee",
               ExcelTemplate.ALL_ROWS);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          Dim Conn As New OleDbConnection()
          Dim EmployeeDt As DataTable = Nothing
          Try
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL Query for employee information
               Dim EmployeeSQL As String = "SELECT FirstName + ' ' + " & _
                    "LastName As Name, Title " & _
                    "FROM Employees WHERE EmployeeID=?"
               Dim CmdEmployee As New OleDbCommand(EmployeeSQL, Conn)
               CmdEmployee.Parameters.Add("@EmployeeID", EmployeeId)
               Dim AdptEmployee As New OleDbDataAdapter(CmdEmployee)
               EmployeeDt = New DataTable()
               AdptEmployee.Fill(EmployeeDt)
          End Try
          xlt.SetDataSource(EmployeeDt, _
               "Employee", _
               ExcelTemplate.ALL_ROWS)
```

ExcelTemplate.SetDataSource(System.Data.DataTable, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(System.Data.DataTable, String, DataBindingProperties) method.

Description

Sets an ADO.NET DataTable as a data source to bind to template data markers.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public virtual void SetDataSource(System.Data.DataTable source, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Overridable Sub SetDataSource(ByVal source As System.Data.DataTable, ByVal dataSourceName As
String)
```

Parameters

source

The DataTable to use as the data source.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          OleDbConnection Conn = new OleDbConnection();
          DataTable EmployeeDt = null;
          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);
          }
          xlt.SetDataSource(EmployeeDt, "Employee");
```

vb.net

```
Dim xlt As New ExcelTemplate()
          Dim Conn As New OleDbConnection()
          Dim EmployeeDt As DataTable = Nothing
          Try
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL Query for employee information
               Dim EmployeeSQL As String = "SELECT FirstName + ' ' + " & _
                    "LastName As Name, Title " & _
                    "FROM Employees WHERE EmployeeID=?"
               Dim CmdEmployee As New OleDbCommand(EmployeeSQL, Conn)
               CmdEmployee.Parameters.Add("@EmployeeID", EmployeeId)
               Dim AdptEmployee As New OleDbDataAdapter(CmdEmployee)
               EmployeeDt = New DataTable()
               AdptEmployee.Fill(EmployeeDt)
          End Try
          xlt.SetDataSource(EmployeeDt, "Employee")
```

ExcelTemplate.SetDataSource(System.Data.IDataReader, String, Int32)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(System.Data.IDataReader, String, DataBindingProperties) method.

Description

Sets an IDataReader as a data source to bind to template data markers.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public void SetDataSource(System.Data.IDataReader source, System.String dataSourceName, int maxRows)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Sub SetDataSource(ByVal source As System.Data.IDataReader, ByVal dataSourceName As String,
ByVal maxRows As Integer)
```

Parameters

source

This parameter must be an instance of `System.Data.SqlClient.SqlDataReader`, `System.Data.OleDb.OleDbDataReader` or `Microsoft.AnalysisServices.AdomdClient.AdomdDataReader`, otherwise `SetDataSource` will throw an exception.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

maxRows

Specifies the maximum number of rows to import from the data source. To import the maximum rows available, use the constant ExcelTemplate.ALL_ROWS.

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
OleDbDataReader CategoryRdr = GetCategoryReader();
          try
          {
               ExcelTemplate xlt = new ExcelTemplate();
               xlt.SetDataSource(CategoryRdr, "Category", ExcelTemplate.ALL_ROWS);
          } catch(System.Exception ex) {
          } finally {
               //--- The OleDbDataReader must be open when it's passed
               //--- to ExcelTemplate. Remember to close it after use.
               if(CategoryRdr != null)
                    CategoryRdr.Close();
          }

          //--- Get an OleDbDataReader containing  a list of product categories
          private OleDbDataReader GetCategoryReader()
          {
               OleDbConnection Conn = new OleDbConnection();
               Conn.ConnectionString = Application["connstring"].ToString();

               //--- SQL query for a list of categories
               string CategorySQL = "SELECT CategoryName FROM Categories";
               OleDbCommand Cmd = new OleDbCommand(CategorySQL, Conn);
               Conn.Open();
               return Cmd.ExecuteReader();
          }
```

vb.net

```
Dim CategoryRdr As OleDbDataReader = GetCategoryReader()
          Try
               Dim xlt As New ExcelTemplate()
               xlt.SetDataSource(CategoryRdr, "Category", ExcelTemplate.ALL_ROWS)
          Catch ex System.Exception
          Finally
               '--- The OleDbDataReader must be open when it's passed
               '--- to ExcelTemplate. Remember to close it after use.
               If Not CategoryRdr = Nothing
                    CategoryRdr.Close()
               End If
          End Try

          '--- Get an OleDbDataReader containing a list of product categories
          Private Function GetCategoryReader() As OleDbDataReader
               Dim Conn As New OleDbConnection()
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL query for a list of categories
               Dim CategorySQL As String = "SELECT CategoryName FROM Categories"
               Dim Cmd As New OleDbCommand(CategorySQL, Conn)
               Conn.Open()
               Return Cmd.ExecuteReader()
          End Function
```

ExcelTemplate.SetDataSource(System.Data.IDataReader, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindData(System.Data.IDataReader, String, DataBindingProperties) method.

Description

Sets an ADO.NET IDataReader as a data source to bind to template data markers.

Signature: C#

```
[Obsolete("SetDataSource is deprecated, use BindData instead")]
 public void SetDataSource(System.Data.IDataReader source, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetDataSource is deprecated, use BindData instead")> _
 Public Sub SetDataSource(ByVal source As System.Data.IDataReader, ByVal dataSourceName As String)
```

Parameters

source

This parameter must be an instance of `System.Data.SqlClient.SqlDataReader` or `System.Data.OleDb.OleDbDataReader` or `Microsoft.AnalysisServices.AdomdClient.AdomdDataReader`, otherwise `SetDataSource` will throw an exception.

dataMarkerName

The name of the set of data markers at which to insert the values imported from the data source. `dataMarkerName` must be specified, but can be left as null or an empty string if this is the first data source bound. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

Exceptions

ArgumentNullException

`SetDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
OleDbDataReader CategoryRdr = GetCategoryReader();
          try
          {
               ExcelTemplate xlt = new ExcelTemplate();
               xlt.SetDataSource(CategoryRdr, "Category");
          } catch(System.Exception ex) {
          } finally {
               //--- The OleDbDataReader must be open when it's passed
               //--- to ExcelTemplate. Remember to close it after use.
               if (CategoryRdr != null)
                    CategoryRdr.Close();
          }

          //--- Get an OleDbDataReader containing a list of product categories
          private OleDbDataReader GetCategoryReader()
          {
               OleDbConnection Conn = new OleDbConnection();
               Conn.ConnectionString = Application["connstring"].ToString();

               //--- SQL query for a list of categories
               string CategorySQL = "SELECT CategoryName FROM Categories";
               OleDbCommand Cmd = new OleDbCommand(CategorySQL, Conn);
               Conn.Open();
               return Cmd.ExecuteReader();
          }
```

vb.net

```
Dim CategoryRdr As OleDbDataReader = GetCategoryReader()
          Try
               Dim xlt As New ExcelTemplate()
               xlt.SetDataSource(CategoryRdr, "Category")
          Catch ex As System.Exception
          Finally
               '--- The OleDbDataReader must be open when it's passed
               '--- to ExcelTemplate. Remember to close it after use.
               If Not CategoryRdr = Nothing
                    CategoryRdr.Close()
               End If
          End Try

          '--- Get an OleDbDataReader containing a list of product categories
          Private Function GetCategoryReader() As OleDbDataReader
               Dim Conn As New OleDbConnection()
               Conn.ConnectionString = Application("connstring").ToString()

               '--- SQL query for a list of categories
               Dim CategorySQL As String = "SELECT CategoryName FROM Categories"
               Dim Cmd As New OleDbCommand(CategorySQL, Conn)
               Conn.Open()
               Return Cmd.ExecuteReader()
          End Function
```

ExcelTemplate.SetRowDataSource

Overload	Description
SetRowDataSource(Object(), String(), String)	Sets an array of objects as a data source for a single row in a template.
SetRowDataSource(System.Collections.IDictionary, String)	Sets a row's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.
SetRowDataSource(System.Collections.IEnumerable, String(), String)	Sets a row's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.
SetRowDataSource(System.Data.DataSet, String)	Sets a DataSet as a data source to bind to a row in the template. Only the first row of the DataSet will be used. This method is commonly used to create a set of key-value pairs via a single row of data.
SetRowDataSource(System.Data.DataTable, String)	Sets a DataTable as a data source to bind to a row in the template. Only the first row of the DataTable will be used. This method is commonly used to create a set of key-value pairs via a single row of data.
SetRowDataSource(System.Data.DataView, String)	Sets a DataView as a data source to bind to a row in the template. Only the first row of the DataView will be used. This method is commonly used to create a set of key-value pairs via a single row of data.

ExcelTemplate.SetRowDataSource(Object(), String(), String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindRowData(Object(), String(), String, DataBindingProperties) method.

Description

Sets an array of objects as a data source for a single row in a template.

Signature: C#

```
[Obsolete("SetRowDataSource is deprecated, use BindRowData instead")]
 public void SetRowDataSource(System.Object[] rowData, System.String[] columnNames, System.String
dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetRowDataSource is deprecated, use BindRowData instead")> _
 Public Sub SetRowDataSource(ByVal rowData As Object(), ByVal columnNames As String(), ByVal
dataSourceName As String)
```

Parameters

aData

An object to use as the data source.

aColumnNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `columnNames` is specified, both ordinal field binding and named field binding can be used.

aDataMarkerName

The name of the set of data markers at which to insert the data source value. This parameter must be specified, but can be null or an empty string when setting the first data source to bind to the template. Note: `dataMarkerName` does not include a data marker's column name, for example, the `dataMarkerName` for `%%=Products.ProductID` is "Products."

Exceptions

ArgumentNullException

SetRowDataSource will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Examples

C#

```
//--- Set a data source for the row of data markers
          //--- %%=Address.Street, %%=Address.City, and
          //--- %%=Address.State
          string[] addressvalues = {"1330 Beacon St.", "Brookline", "MA"};
          string[] addressnames = {"Street", "City", "State"};
          xlt.SetRowDataSource(addressvalues, addressnames, "Address");
```

vb.net

```
'--- Set a data source for the row of data markers
          '--- %%=Address.Street, %%=Address.City, and
          '--- %%=Address.State
          Dim addressvalues As String()= {"1330 Beacon St.", "Brookline", "MA"}
          Dim addressnames As String() = {"Street", "City", "State"}
          xlt.SetRowDataSource(addressvalues, addressnames, "Address")
```

ExcelTemplate.SetRowDataSource(System.Collections.IDictionary, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindRowData(System.Collections.IDictionary, String, DataBindingProperties) method.

Description

Sets a row's data source to an `IDictionary`. An `IDictionary` represents a collection of key-and-value pairs.

Signature: C#

```
[Obsolete("SetRowDataSource is deprecated, use BindRowData instead")]
 public void SetRowDataSource(System.Collections.IDictionary dataSource, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetRowDataSource is deprecated, use BindRowData instead")> _
 Public Sub SetRowDataSource(ByVal dataSource As System.Collections.IDictionary, ByVal dataSourceName
As String)
```

Parameters

dataSource

An `IDictionary` collection of key-and-value pairs to use as the data source.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

Exceptions

ArgumentNullException

`SetRowDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the IDictionary collection to the
          //--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersColl, "Orders");
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
ExcelTemplate xlt = new ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the IDictionary collection to the
          '--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersColl, "Orders")
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.SetRowDataSource(System.Collections.IEnumerable, String(), String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindRowData(System.Collections.IEnumerable, String(), String, DataBindingProperties) method.

Description

Sets a row's data source to an `IEnumerable` collection. The `IEnumerable` interface supports a simple iteration over a collection.

Signature: C#

```
[Obsolete("SetRowDataSource is deprecated, use BindRowData instead")]
 public void SetRowDataSource(System.Collections.IEnumerable dataSource, System.String[] columnNames,
System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetRowDataSource is deprecated, use BindRowData instead")> _
 Public Sub SetRowDataSource(ByVal dataSource As System.Collections.IEnumerable, ByVal columnNames As
String(), ByVal dataSourceName As String)
```

Parameters

dataSource

An `IEnumerable` collection to use as the data source.

colNames

The names of the columns to get from the data source. If the `columnNames` parameter is null, field binding can only be performed by ordinal (for example, %%=DSN.#1 or %%=$DSN). If `colNames` is specified, both ordinal field binding and named field binding can be used.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

Exceptions

ArgumentNullException

`SetRowDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the IEnumerable collection to the
          //--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersColl, null,"Orders");
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the IEnumerable collection to the
          '--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersColl, null,"Orders")
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.SetRowDataSource(System.Data.DataSet, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindRowData(System.Data.DataSet, String, DataBindingProperties) method.

Description

Sets a DataSet as a data source to bind to a row in the template. Only the first row of the DataSet will be used. This method is commonly used to create a set of key-value pairs via a single row of data.

Signature: C#

```
[Obsolete("SetRowDataSource is deprecated, use BindRowData instead")]
 public void SetRowDataSource(System.Data.DataSet dataSource, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetRowDataSource is deprecated, use BindRowData instead")> _
 Public Sub SetRowDataSource(ByVal dataSource As System.Data.DataSet, ByVal dataSourceName As String)
```

Parameters

source

The DataSet to use as the data source.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

Exceptions

ArgumentNullException

`SetRowDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the DataSet to the
          //--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersDS, "Orders");
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the DataSet to the
          '--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersDS, "Orders")
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.SetRowDataSource(System.Data.DataTable, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindRowData(System.Data.DataTable, String, DataBindingProperties) method.

Description

Sets a DataTable as a data source to bind to a row in the template. Only the first row of the DataTable will be used. This method is commonly used to create a set of key-value pairs via a single row of data.

Signature: C#

```
[Obsolete("SetRowDataSource is deprecated, use BindRowData instead")]
 public void SetRowDataSource(System.Data.DataTable dataSource, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetRowDataSource is deprecated, use BindRowData instead")> _
 Public Sub SetRowDataSource(ByVal dataSource As System.Data.DataTable, ByVal dataSourceName As
String)
```

Parameters

source

The DataTable to use as the data source.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

Exceptions

ArgumentNullException

`SetRowDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the DataTable to the
          //--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersDT, "Orders");
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the DataTable to the
          '--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersDT, "Orders")
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.SetRowDataSource(System.Data.DataView, String)

This method has been deprecated in version 7.5. Please use the ExcelTemplate.BindRowData(System.Data.DataView, String, DataBindingProperties) method.

Description

Sets a DataView as a data source to bind to a row in the template. Only the first row of the DataView will be used. This method is commonly used to create a set of key-value pairs via a single row of data.

Signature: C#

```
[Obsolete("SetRowDataSource is deprecated, use BindRowData instead")]
 public void SetRowDataSource(System.Data.DataView dataSource, System.String dataSourceName)
```

Signature: VB.NET

```
<Obsolete("SetRowDataSource is deprecated, use BindRowData instead")> _
 Public Sub SetRowDataSource(ByVal dataSource As System.Data.DataView, ByVal dataSourceName As String)
```

Parameters

source

The DataView to use as the data source.

dataSourceName

The name of the data marker at which to insert the values imported from the data source. For example, to bind a data source to the data marker %%=Orders.OrderId, the value of dataSourceName should be "Orders". The string passed must begin with a letter.

Exceptions

ArgumentNullException

`SetRowDataSource` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: SetCellDataSource, SetColumnDataSource, SetRowDataSource, and SetDataSource.

Examples

C#

```
ExcelTemplate xlt = new ExcelTemplate();
          xlt.Open(@"C:\ExcelWriter\EmployeeOrdersTemplate.xls");

          //--- Bind the DataView to the
          //--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersDV, "Orders");
          xlt.Process();
          xlt.Save(Page.Response, "EmployeeOrders.xls", false);
```

vb.net

```
Dim xlt As New ExcelTemplate()
          xlt.Open("C:\ExcelWriter\EmployeeOrdersTemplate.xls")

          '--- Bind the DataView to the
          '--- %%=Orders.[Field] data marker.
          xlt.SetRowDataSource(OrdersDV, "Orders")
          xlt.Process()
          xlt.Save(Page.Response, "EmployeeOrders.xls", False)
```

ExcelTemplate.StretchCellReferencesInCharts

Description

Sets or returns whether to stretch cell references in charts.

Signature: C#

```
public boolean StretchCellReferencesInCharts{ get; set; }
```

Signature: VB.NET

```
Public Property StretchCellReferencesInCharts() As Boolean
```

Examples

C#

```
//--- Get StretchCellReferencesInCharts
          boolean stretchCellRef = XlwTemplate.StretchCellReferencesInCharts;

          //--- Set StretchCellReferencesInCharts to true
          XlwTemplate.StretchCellReferencesInCharts = true;
```

vb.net

```
'--- Get StretchCellReferencesInCharts
          Dim stretchCellRef As Boolean = XlwTemplate.StretchCellReferencesInCharts

          '--- Set StretchCellReferencesInCharts to true
          XlwTemplate.StretchCellReferencesInCharts = True
```

ExcelTemplate.Version

Description

Returns the exact version of ExcelWriter.

Signature: C#

```
public System.String Version{ get; }
```

Signature: VB.NET

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

Remarks

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

PlaceHolder	Description
ProductName	A product name like "SoftArtisans.ExcelWriter." 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.

Version is a **Read Only** property.

Examples

C#

```
string version = xlt.Version;

          //--- Bind the version to the data marker
          //--- %%=$ExcelWriterVersion.
          xlt.SetCellDataSource(version, "ExcelWriterVersion");
```

vb.net

```
Dim version As String = xlt.Version

          '--- Bind the version to the data marker
          '--- %%=$ExcelWriterVersion.
          xlt.SetCellDataSource(version, "ExcelWriterVersion")
```

SharePointIntegration Programmer's Reference

Introduced in build 7.1.0.1855

SharePointIntegration

Introduced in build 7.1.0.1855

Description

The SharePointIntegration class contains the extension methods that allow users to bind SharePoint data sources to Excel spreadsheets 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 ExcelTemplate, ExcelApplication, Worksheet, and Area objects with SharePoint. SharePointIntegration belongs to the SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration namespace, which has the Microsoft.SharePoint class as a dependency. Additionally, the SharePointIntegration class belongs in a separate DLL, SoftArtisans.OfficeWriter.ExcelWriter.SharePoinIntegration.dll, which must be added to the bin directory of your project.

SharePointIntegration is a static class which cannot directly be instantiated, but rather contains extension methods for the ExcelApplication, ExcelTemplate, Worksheet, and Area objects. Use SharePointIntegration extensions just as you would the other methods of each object, remembering to add references to the SoftArtisans.OfficeWriter.ExcelWriter and Microsoft.SharePoint namespaces.

Methods

Name	Description
BindData(ExcelTemplate, Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint List as a template data source.
BindData(ExcelTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint View as a template data source.
BindRowData(ExcelTemplate, Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint List as a data source for a single row in a template.

BindRowData(ExcelTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties)	Sets a SharePoint View as a data source for a single row in a template.
ImportData(Area, Microsoft.SharePoint.SPList)	Imports data from a SharePoint List to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Area, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint List to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Area, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint View to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Area, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Imports data from a SharePoint View to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Worksheet, Cell, Microsoft.SharePoint.SPList)	Imports data from a SharePoint List to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Worksheet, Cell, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint List to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Worksheet, Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)	Imports data from a SharePoint View to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
ImportData(Worksheet, Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)	Imports data from a SharePoint View to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.
Open(ExcelApplication, Microsoft.SharePoint.SPDocumentLibrary, String)	Opens an existing Excel spreadsheet from a SharePoint Document Library.
Open(ExcelApplication, Microsoft.SharePoint.SPListItem, String)	Opens an existing Excel spreadsheet that is attached to a SharePoint List Item.
Open(ExcelTemplate, Microsoft.SharePoint.SPDocumentLibrary, String)	Opens a spreadsheet from a SharePoint Document Library.
Open(ExcelTemplate, Microsoft.SharePoint.SPListItem, String)	Opens a spreadsheet from a SharePoint List Item.
Save(ExcelApplication, Workbook, Microsoft.SharePoint.SPListItem, String)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.
Save(ExcelApplication, Workbook, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.
Save(ExcelTemplate, Microsoft.SharePoint.SPListItem, String)	Generates an Excel binary or OOXML file and saves it as an attachment to a SharePoint list item.
Save(ExcelTemplate, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)	Generates an Excel binary or OOXML file and saves it to a SharePoint Document Library.

SharePointIntegration.BindData(ExcelTemplate, Microsoft.SharePoint.SPList, String, DataBindingProperties)

Introduced in build 7.1.0.1855

Description

Sets a SharePoint List as a template data source.

Signature: C#

```
public static void BindData(this ExcelTemplate template, Microsoft.SharePoint.SPList view, String
dataSourceName, DataBindingProperties properties)
```

Signature: VB.NET

```
Public Shared Sub BindData(ByVal template As ExcelTemplate, ByVal list As Microsoft.SharePoint.SPList,
ByVal dataSourceName As String, ByVal properties As DataBindingProperties)
```

Parameters

template

The current ExcelTemplate object which is calling BindData

list

SharePoint List to use as a data source for the template

dataSourceName

Name of the data source in the template

properties

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

> This is an extension method for the ExcelTemplate object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

SharePointIntegration.BindData(ExcelTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties)

Introduced in build 7.1.0.1855

Description

Sets a SharePoint View as a template data source.

Signature: C#

```
public static void BindData(this ExcelTemplate template, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, String dataSourceName, DataBindingProperties properties)
```

Signature: VB.NET

```
Public Shared Sub BindData(ByVal template As ExcelTemplate, ByVal view As Microsoft.SharePoint.SPView,
ByVal list As Microsoft.SharePoint.SPList, ByVal dataSourceName As String, ByVal properties As
DataBindingProperties)
```

Parameters

template

The current ExcelTemplate object which is calling BindData

view

SharePoint View to use as a data source for the template.

list

SharePoint List associated with the SharePoint View

dataSourceName

Name of the data source in the template.

properties

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

You can set several data sources for a single template. Use the following methods to set template data sources: BindCellData, BindColumnData, BindRowData, and BindData.

> This is an extension method for the ExcelTemplate object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

SharePointIntegration.BindRowData(ExcelTemplate, Microsoft.SharePoint.SPList, String, DataBindingProperties)

Introduced in build 7.1.0.1855

Description

Sets a SharePoint List as a data source for a single row in a template.

Signature: C#

```
public static void BindRowData(this ExcelTemplate template, Microsoft.SharePoint.SPList list, String
dataSourceName, DataBindingProperties properties)
```

Signature: VB.NET

```
Public Shared Sub BindRowData(ByVal template As ExcelTemplate, ByVal list as
Microsoft.SharePoint.SPList, ByVal dataSourceName As String, ByVal properties As
DataBindingProperties)
```

Parameters

template

The current ExcelTemplate object which is calling BindRowData

list

SharePoint List to use as a data source for the template

dataSourceName

Name of the data source in the template

properties

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindRowData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

This is an extension method for the ExcelTemplate object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

SharePointIntegration.BindRowData(ExcelTemplate, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, String, DataBindingProperties)

Introduced in build 7.1.0.1855

Description

Sets a SharePoint View as a data source for a single row in a template.

Signature: C#

```
public static void BindRowData(this ExcelTemplate template, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, String dataSourceName, DataBindingProperties properties)
```

Signature: VB.NET

```
Public Shared Sub BindRowData(ByVal template as ExcelTemplate, ByVal view As
Microsoft.SharePoint.SPView, ByVal list As Microsoft.SharePoint.SPList, ByVal dataSourceName As
String, ByVal properties As DataBindingProperties)
```

Parameters

template

The current ExcelTemplate object which is calling BindRowData

view

SharePoint View to use as a data source for the template

list

SharePoint List associated with the SharePoint View

dataSourceName

Name of the data source in the template

properties

The DataBindingProperties object which contains information about how the data should be bound to the template. `property` Must be specified, but the `DataBindingProperties` need not be set beforehand. To bind data to a template with the default `DataBindingProperties`, pass in `ExcelTemplate.CreateDataBindingProperties()` as the `property` value. Otherwise, use the `ExcelTemplate.CreateDataBindingProperties()` method to generate a new `DataBindingProperties` object and set the DataBindingProperties.MaxRows, DataBindingProperties.Transpose, and/or DataBindingProperties.WorksheetName properties for the workbook.

Exceptions

ArgumentNullException

`BindRowData` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

This is an extension method for the ExcelTemplate object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

SharePointIntegration.ImportData(Area, Microsoft.SharePoint.SPList, DataImportProperties)

Introduced in build 7.1.0.1855

Description

Imports data from a SharePoint List to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public static Area ImportData(this Area area, Microsoft.SharePoint.SPList list, DataImportProperties
props)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal area As Area, ByVal list As Microsoft.SharePoint.SPList, ByVal
props As DataImportProperties) As Area
```

Parameters

area

The current Area object which is calling ImportData

list

SharePoint List to use as a data source for the Area

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An `Area` object representing the set of cells populated with the imported values.

Remarks

> This is an extension method for the Area object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
DataImportProperties importProps = wb.CreateDataImportProperties();
          importProps.Transpose = true;
          Area importedArea = a.ImportData(myList, importProps);
```

vb.net

```
Dim importProps As DataImportProperties = wb.CreateDataImportProperties()
          importProps.Transpose = True
          Dim importedArea As Area = a.ImportData(myList, importProps)
```

SharePointIntegration.ImportData(Area, Microsoft.SharePoint.SPList)

Introduced in build 7.1.0.1855

Description

Imports data from a SharePoint List to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public static Area ImportData(this Area area, Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal area As Area, ByVal list As Microsoft.SharePoint.SPList) As
Area
```

Parameters

area

The current Area object which is calling ImportData

list

SharePoint List to use as a data source for the Area

Returns

An `Area` object representing the set of cells populated with the imported values.

Remarks

> This is an extension method for the Area object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
Area importedArea = a.ImportData(myList);
```

vb.net

```
Dim importedArea As Area = a.ImportData(myList)
```

SharePointIntegration.ImportData(Area, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)

Introduced in build 7.1.0.1855

Description

Imports data from a SharePoint View to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public static Area ImportData(this Area area, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, DataImportProperties props)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal area As Area, ByVal view As Microsoft.SharePoint.SPView, ByVal
list As Microsoft.SharePoint.SPList, ByVal props As DataImportProperties) As Area
```

Parameters

area

The current Area object which is calling ImportData

view

SharePoint View to use as a data source for the template

list

SharePoint List associated with the SharePoint View

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An `Area` object representing the set of cells populated with the imported values.

Remarks

This is an extension method for the Area object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
DataImportProperties importProps = wb.CreateDataImportProperties();
          importProps.Transpose = true;
          Area importedArea = a.ImportData(myView, myList, importProps);
```

vb.net

```
Dim importProps As DataImportProperties = wb.CreateDataImportProperties()
          importProps.Transpose = True
          Dim importedArea As Area = a.ImportData(myView, myList, importProps)
```

SharePointIntegration.ImportData(Area, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)

Introduced in build 7.1.0.1855

Description

Imports data from a SharePoint View to the specified Area. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public static Area ImportData(this Area area, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal area As Area, ByVal view As Microsoft.SharePoint.SPView, ByVal
list As Microsoft.SharePoint.SPList) As Area
```

Parameters

area

The current Area object which is calling ImportData

view

SharePoint View to use as a data source for the Area

list

SharePoint List associated with the SharePoint View

Returns

An `Area` object representing the set of cells populated with the imported values.

Remarks

This is an extension method for the Area object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
Area importedArea = a.ImportData(myView, myList);
```

vb.net

```
Dim importedArea As Area = a.ImportData(myView, myList)
```

SharePointIntegration.ImportData(Worksheet, Cell, Microsoft.SharePoint.SPList, DataImportProperties)

Introduced in build 7.1.0.1855

Description

Imports data from a SharePoint List to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public static Area ImportData(this Worksheet worksheet, Cell cell, Microsoft.SharePoint.SPList list,
DataImportProperties props)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal worksheet As Worksheet, ByVal cell As Cell, ByVal list As
Microsoft.SharePoint.SPList, ByVal props As DataImportProperties) As Area
```

Parameters

worksheet

The current Worksheet object which is calling ImportData

cell

The cell at which to start entering the imported values.

list

SharePoint List to use as a data source for the Area starting at the given cell

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An Area object representing the set of cells populated with the imported values.

Remarks

This is an extension method for the Worksheet object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
Area importedArea = ws.ImportData(cell, myList, importProps);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(cell, myList, importProps)
```

SharePointIntegration.ImportData(Worksheet, Cell, Microsoft.SharePoint.SPList)

Introduced in build 7.1.0.1855

Description

Imports data from a SharePoint List to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public static Area ImportData(this Worksheet worksheet, Cell cell, Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal worksheet As Worksheet, ByVal cell As Cell, ByVal list As
Microsoft.SharePoint.SPList) As Area
```

Parameters

worksheet

The current Worksheet object which is calling ImportData

cell

The cell at which to start entering the imported values.

list

SharePoint List to use as a data source for the Area starting at the given cell

Returns

An Area object representing the set of cells populated with the imported values.

Remarks

This is an extension method for the Worksheet object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
Area importedArea = ws.ImportData(cell, myList);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(cell, myList)
```

SharePointIntegration.ImportData(Worksheet, Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList, DataImportProperties)

Introduced in build 7.1.0.1855

Description

Imports data from a SharePoint View to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public static Area ImportData(this Worksheet worksheet, Cell cell, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list, DataImportProperties props)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal worksheet As Worksheet, ByVal cell As Cell, ByVal view As
Microsoft.SharePoint.SPView, ByVal list As Microsoft.SharePoint.SPList, ByVal props As
DataImportProperties) As Area
```

Parameters

worksheet

The current Worksheet object which is calling ImportData

cell

The cell at which to start entering the imported values.

view

SharePoint View to use as a data source for the Area starting at the given cell

list

SharePoint List associated with the SharePoint View

props

A DataImportProperties object that contains a set of properties that will determine the behavior of the data import.

Returns

An Area object representing the set of cells populated with the imported values.

Remarks

This is an extension method for the Worksheet object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
Area importedArea = ws.ImportData(cell, myView, myList, importProps);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(cell, myView, myList, importProps)
```

SharePointIntegration.ImportData(Worksheet, Cell, Microsoft.SharePoint.SPView, Microsoft.SharePoint.SPList)

Introduced in build 7.1.0.1855

Description

Imports data from a SharePoint View to cells in the worksheet. The new data will overwrite values and formulas in the target worksheet cells, but existing formatting will be preserved.

Signature: C#

```
public static Area ImportData(this Worksheet worksheet, Cell cell, Microsoft.SharePoint.SPView view,
Microsoft.SharePoint.SPList list)
```

Signature: VB.NET

```
Public Shared Function ImportData(ByVal worksheet As Worksheet, ByVal cell As Cell, ByVal view As
Microsoft.SharePoint.SPView, ByVal list As Microsoft.SharePoint.SPList) As Area
```

Parameters

worksheet

The current Worksheet object which is calling ImportData

cell

The cell at which to start entering the imported values.

view

SharePoint View to use as a data source for the Area starting at the given cell

list

SharePoint List associated with the SharePoint View

Returns

An Area object representing the set of cells populated with the imported values.

Remarks

This is an extension method for the Worksheet object to be used for binding SharePoint data to spreadsheets from within SharePoint. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
Area importedArea = ws.ImportData(cell, myView, myList);
```

vb.net

```
Dim importedArea As Area = ws.ImportData(cell, myView, myList)
```

SharePointIntegration.Open(ExcelApplication, Microsoft.SharePoint.SPDocumentLibrary, String)

Introduced in build 7.1.0.1855

Description

Opens an existing Excel spreadsheet from a SharePoint Document Library.

Signature: C#

```
public static Workbook Open(this ExcelApplication excelApplication,
Microsoft.SharePoint.SPDocumentLibrary docLib, String fileName)
```

Signature: VB.NET

```
Public Shared Function Open(ByVal excelApplication As ExcelApplication, ByVal docLib As
Microsoft.SharePoint.SPDocumentLibrary, ByVal fileName As String) As Workbook
```

Parameters

excelApplication

The current ExcelApplication object which is opening the file

docLib

SharePoint Document Library that contains the Excel spreadsheet being opened

fileName

The name of the file to open. The file must be a BIFF8 format (Excel 97 or later) .xls or .xlt file.

Returns

A Workbook object representing the file opened.

Exceptions

System.Exception

If the file cannot be found or opened, or if ExcelWriter cannot parse the file as a BIFF8 file.

Remarks

ExcelWriter supports Excel's BIFF8 (Excel 97 or later) format only. Do not use this method to open BIFF7 (Excel 95) files.

> This is an extension method for the ExcelApplication object to be used for opening spreadsheets from SharePoint Document Libraries. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
Workbook wb = xla.Open(myDocLib, "Report.xls");
```

vb.net

```
Dim wb As Workbook = xla.Open(myDocLib, "Report.xls")
```

SharePointIntegration.Open(ExcelApplication, Microsoft.SharePoint.SPListItem, String)

Introduced in build 7.1.0.1855

Description

Opens an existing Excel spreadsheet that is attached to a SharePoint List Item.

Signature: C#

```
public static Workbook Open(this ExcelApplication excelApplication, Microsoft.SharePoint.SPListItem
listItem, String fileName)
```

Signature: VB.NET

```
Public Shared Function Open(ByVal excelApplication As ExcelApplication, ByVal listItem As
Microsoft.SharePoint.SPListItem, ByVal fileName As String) As Workbook
```

Parameters

excelApplication

The current ExcelApplication object which is opening the file

listItem

SharePoint List Item to which the file is attached

fileName

The name of the file to open. The file must be a BIFF8 format (Excel 97 or later) .xls or .xlt file.

Returns

A Workbook object representing the file opened.

Exceptions

System.Exception

If the file cannot be found or opened, or if ExcelWriter cannot parse the file as a BIFF8 file.

Remarks

ExcelWriter supports Excel's BIFF8 (Excel 97 or later) format only. Do not use this method to open BIFF7 (Excel 95) files.

> This is an extension method for the ExcelApplication object to be used for opening spreadsheets from SharePoint List Items. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

Examples

C#

```
Workbook wb = xla.Open(myListItem, "Report.xls");
```

vb.net

```
Dim wb As Workbook = xla.Open(myListItem, "Report.xls")
```

SharePointIntegration.Open(ExcelTemplate, Microsoft.SharePoint.SPDocumentLibrary, String)

Introduced in build 7.1.0.1855

Description

Opens a spreadsheet from a SharePoint Document Library.

Signature: C#

```
public static void Open(this ExcelTemplate template, Microsoft.SharePoint.SPDocumentLibrary docLib,
String fileName)
```

Signature: VB.NET

```
Public Shared Sub Open(ByVal template As ExcelTemplate, ByVal docLib As
Microsoft.SharePoint.SPDocumentLibrary, ByVal fileName As String)
```

Parameters

template

The current ExcelTemplate object which is opening the file

docLib

SharePoint Document Library that contains the Excel spreadsheet being opened

fileName

The name of the file to open. The file must be a BIFF8 format (Excel 97 or later) .xls or .xlt file or an OOXML format .xlsx, .xlsm, .xltx, or .xltm file.

Exceptions

ArgumentNullException

Open will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

> This is an extension method for the ExcelTemplate object to be used for opening spreadsheets from SharePoint Document Libraries. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

SharePointIntegration.Open(ExcelTemplate, Microsoft.SharePoint.SPListItem, String)

Introduced in build 7.1.0.1855

Description

Opens a spreadsheet from a SharePoint List Item.

Signature: C#

```
public static void Open(this ExcelTemplate template, Microsoft.SharePoint.SPListItem listItem, String
fileName)
```

Signature: VB.NET

```
Public Shared Sub Open(ByVal template As ExcelTemplate, ByVal listItem As
Microsoft.SharePoint.SPListItem, ByVal fileName As String)
```

Parameters

template

The current ExcelTemplate object which is opening the file

listItem

SharePoint List Item to which the file is attached

fileName

The name of the file to open. The file must be a BIFF8 format (Excel 97 or later) .xls or .xlt file or an OOXML format .xlsx, .xlsm, .xltx, or .xltm file.

Exceptions

ArgumentNullException

Open will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

This is an extension method for the ExcelTemplate object to be used for opening spreadsheets from SharePoint List Items. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.SharePointIntegration.dll

SharePointIntegration.Save(ExcelApplication, Workbook, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)

Introduced in build 7.1.0.1855

Description

Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.

Signature: C#

```
public static void Save(this ExcelApplication excelApplication, Workbook workbook,
Microsoft.SharePoint.SPDocumentLibrary docLib, String fileName, bool overwrite)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal excelApplication As ExcelApplication, ByVal workbook As Workbook,
ByVal docLib As Microsoft.SharePoint.SPDocumentLibrary, ByVal fileName As String, ByVal overwrite As
Boolean)
```

Parameters

excelApplication

The current ExcelApplication object which is saving the workbook to file

workbook

A `Workbook` object representing the workbook to save.

docLib

SharePoint Document Library that contains the Excel spreadsheet being opened

fileName

Specifies a file name for the Workbook. ExcelWriter will save the file to the document library using this name. If a file with the same name exists, it will be overwritten by the new Excel file if the overwrite parameter is set to True.

overwrite

Set to True to overwrite an Excel file if one already exists by that name in the Document Library

Exceptions

System.Exception

If there is a problem creating, opening, or writing to the file specified, or reading from the workbook object.

Remarks

This is an extension method for the ExcelApplication object to be used for saving spreadsheets to SharePoint Document Libraries. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.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;
xla.Save(wb, documentLibrary, "populated.xls", 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
xla.Save(wb, documentLibrary, "populated.xls", True)

'--- Update the document library
documentLibrary.Update()

'--- Return the AllowUnsafeUpdates value to its original state
documentLibrary.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates
```

SharePointIntegration.Save(ExcelApplication, Workbook, Microsoft.SharePoint.SPListItem, String)

Introduced in build 7.1.0.1855

Description

Saves a specified Workbook as a BIFF8 format (Excel 97 or later) .xls file.

Signature: C#

```
public static void Save(this ExcelApplication excelApplication, Workbook workbook,
Microsoft.SharePoint.SPListItem listItem, String fileName)
```

Signature: VB.NET

```
Public Overridable Sub Save(ByVal excelApplication As ExcelApplication, ByVal workbook As Workbook,
ByVal listItem As Microsoft.SharePoint.SPListItem, ByVal fileName As String)
```

Parameters

excelApplication

The current ExcelApplication object which is saving the workbook to file

workbook

A `Workbook` object representing the workbook to save.

listItem

SharePoint List Item to which the file is attached

fileName

Specifies a file name for the Workbook. ExcelWriter will save the file as an attachment to the List Item using this name.

Exceptions

System.Exception

If there is a problem creating, opening, or writing to the file specified, or reading from the workbook object.

Remarks

> This is an extension method for the ExcelApplication object to be used for saving spreadsheets to SharePoint List Items. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.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;
xla.Save(wb, listItem, "populated.xls");

//--- 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
xla.Save(wb, listItem, "populated.xls")

'--- Update the list
listItem.Update()

'--- Return the AllowUnsafeUpdates value to its original state
list.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates
```

SharePointIntegration.Save(ExcelTemplate, Microsoft.SharePoint.SPDocumentLibrary, String, Boolean)

Introduced in build 7.1.0.1855

Description

Generates an Excel binary or OOXML file and saves it to a SharePoint Document Library.

Signature: C#

```
public static void Save(this ExcelTemplate template, Microsoft.SharePoint.SPDocumentLibrary docLib,
string fileName, bool overwrite)
```

Signature: VB.NET

```
Public Shared Sub Save(ByVal template As ExcelTemplate, ByVal docLib As
Microsoft.SharePoint.SPDocumentLibrary, ByVal fileName As String, ByVal overwrite As Boolean)
```

Parameters

template

The current ExcelTemplate object which is being saved.

docLib

SharePoint Document Library that contains the Excel spreadsheet being opened

fileName

Name of the output file. ExcelWriter will save the file to the document library using this name. If a file with the same name exists, it will be overwritten by the new Excel file if the overwrite parameter is set to True.

overwrite

Set to True to overwrite an Excel file if one already exists by that name in the Document Library

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

ExcelWriter allows you to save in both the Excel 97-03 BIFF8 format (.xls) or the new Office Open XML (.xlsx) format. The template file must be of the expected output format. Hence, if you wish to output .xls files, you must start with a .xls template, and if you wish to output .xlsx files, you must start with a .xlsx template. ExcelWriter does not support the creation or editing of .xlsx files with the ExcelApplication object.

You can call `Save` more than once for a single instance of ExcelTemplate. 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 ExcelTemplate object to be used for saving spreadsheets to SharePoint Document Libraries. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.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;
xlt.Save(documentLibrary, "populated.xlsx", 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
xlt.Save(documentLibrary, "populated.xlsx", True)

'--- Update the document library
documentLibrary.Update()

'--- Return the AllowUnsafeUpdates value to its original state
documentLibrary.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates
```

SharePointIntegration.Save(ExcelTemplate, Microsoft.SharePoint.SPListItem, String)

Introduced in build 7.1.0.1855

Description

Generates an Excel binary or OOXML file and saves it as an attachment to a SharePoint list item.

Signature: C#

```
public static void Save(this ExcelTemplate template, Microsoft.SharePoint.SPListItem listItem, string
fileName)
```

Signature: VB.NET

```
Public Shared Sub Save(ByVal template As ExcelTemplate, ByVal listItem As
Microsoft.SharePoint.SPListItem, ByVal fileName As String)
```

Parameters

template

The current ExcelTemplate object which is being saved.

listItem

SharePoint List Item to which the file is attached

fileName

Name of the output file. ExcelWriter will save the file to the list item using this name.

Exceptions

ArgumentNullException

`Save` will throw this exception if `null` (C#) or `Nothing` (VB.NET) is passed to the method.

ArgumentException

Remarks

ExcelWriter allows you to save in both the Excel 97-03 BIFF8 format (.xls) or the new Office Open XML (.xlsx) format. The template file must be of the expected output format. Hence, if you wish to output .xls files, you must start with a .xls template, and if you wish to output .xlsx files, you must start with a .xlsx template. ExcelWriter does not support the creation or editing of .xlsx files with the ExcelApplication object.

You can call `Save` more than once for a single instance of ExcelTemplate. 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 ExcelTemplate object to be used for saving spreadsheets to SharePoint List Items. To use this method, you must add a reference to SoftArtisans.OfficeWriter.ExcelWriter.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;
xlt.Save(listItem, "populated.xlsx");

//--- 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
xlt.Save(listItem, "populated.xlsx")

'--- Update the list
listItem.Update()

'--- Return the AllowUnsafeUpdates value to its original state
list.ParentWeb.AllowUnsafeUpdates = currentAllowUnsafeUpdates
```

OfficeWriter Links

Documentation for other parts of OfficeWriter

In addition to ExcelWriter, OfficeWriter includes the following products:

- **WordWriter**
  - **WordWriter** allows you to create Word documents 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 for OfficeWriter with the option to run and view code:

http://windemo.softartisans.com/OfficeWriter/latest/aspnet.aspx

External Links

- OfficeWriter Online
- OfficeWriter Forums
- SoftArtisans Support and Services
- SoftArtisans
- General Questions