1. ExcelWriter COM for Windows Version 7 . . . 8
1.1 Introduction . . . 8
1.1.1 The Web Reporting Solution . . . 8
1.1.2 What is ExcelWriter? . . . 9
1.1.3 Features and Benefits . . . 10
1.1.4 ExcelWriter Editions . . . 10
1.1.5 Requirements . . . 11
1.1.6 Frequently Asked Questions . . . 11
1.1.7 Change Log . . . 12
1.2 Quick Start . . . 13
1.2.1 Create a Spreadsheet with ExcelTemplate . . . 13
1.2.2 Create a Spreadsheet with ExcelApplication . . . 14
1.3 ExcelTemplate in Depth . . . 16
1.3.1 How to Use Templates . . . 16
1.3.2 Creating Data Markers . . . 17
1.3.3 Using an Array as a Data Source . . . 20
1.3.4 Using a Database as a Data Source . . . 23
1.3.5 Output Options . . . 24
1.3.6 Templates and Charts . . . 24
1.3.7 Templates and PivotTables . . . 28
1.3.8 Passing ExcelTemplate to ExcelApplication . . . 29
1.4 ExcelApplication in Depth . . . 30
1.4.1 Addressing Cells . . . 31
1.4.2 Setting Values . . . 33
1.4.3 Adding a Formula . . . 35
1.4.4 Adding Formatting to a Spreadsheet . . . 37
1.4.5 ExcelApplication Output Options . . . 40
1.4.6 Preserving Leading Zeros in Numeric Strings . . . 41
1.4.7 Reading an Existing Spreadsheet . . . 42
1.4.8 Modifying an Existing Spreadsheet . . . 43
1.4.9 Importing Data from a Database . . . 47
1.4.10 Adding Charts . . . 48
1.4.11 Creating a Complex Chart . . . 50
1.4.12 Creating a PivotTable . . . 53
1.4.13 The Range Object . . . 56
1.4.14 Page Setup . . . 60
1.4.15 Formatting Headers and Footers . . . 63
1.4.16 Protecting Your Worksheet . . . 65
1.4.17 Multilingual Support . . . 67
1.4.18 Passing ExcelApplication to ExcelTemplate . . . 68
1.5 HotCell Technology . . . 69
1.6 Programmer's Reference . . . 77
1.6.1 Object Model . . . 83
1.6.2 ExcelTemplate Programmer's Reference . . . 84
1.6.2.1 ExcelTemplate . . . 84
1.6.2.1.1 ExcelTemplate.AnsiToUnicode(String, Long) . . . 85
1.6.2.1.2 ExcelTemplate.ContentType . . . 86
1.6.2.1.3 ExcelTemplate.DataSource . . . 86
1.6.2.1.4 ExcelTemplate.DecryptPassword . . . 90
1.6.2.1.5 ExcelTemplate.EncryptPassword . . . 90
1.6.2.1.6 ExcelTemplate.Open(String) . . . 91
1.6.2.1.7 ExcelTemplate.PreserveStrings . . . 91
1.6.2.1.8 ExcelTemplate.Process(String, SAProcessMethod, Long) . . . 92
1.6.2.1.9 ExcelTemplate.SpreadsheetsCreatedToday . . . 92
1.6.2.1.10 ExcelTemplate.UnicodeToAnsi(String, Long) . . . 93
1.6.3 ExcelApplication Programmer's Reference . . . 93
1.6.3.1 3DProperties . . . 94
1.6.3.1.1 3DProperties.ChartDepth . . . 95
1.6.3.1.2 3DProperties.DistanceFromEyeToChart . . . 95
1.6.3.1.3 3DProperties.ElevationAngle . . . 96
1.6.3.1.4 3DProperties.PlotVolumeHeight . . . 96
1.6.3.1.5 3DProperties.RotationAngle . . . 97
1.6.3.1.6 3DProperties.SeriesGapDepth . . . 97
1.6.3.1.7 3DProperties.Show3D . . . 98
1.6.3.1.8 3DProperties.Stack3DColumns . . . 98
1.6.3.1.9 3DProperties.UseAutoScaling . . . 99
1.6.3.1.10 3DProperties.UsePerspectiveTransform . . . 99
1.6.3.1.11 3DProperties.ZAxis . . . 100
1.6.3.2 Area . . . 100
1.6.3.2.1 Area.AreaFormatting . . . 101
1.6.3.2.2 Area.BackgroundColor . . . 101
1.6.3.2.3 Area.ForegroundColor . . . 102
1.6.3.2.4 Area.Pattern . . . 102
1.6.3.3 Axis . . . 103
1.6.3.3.1 Axis.AxisCrossesMidCategory . . . 105

1.6.3.3.2 Axis.AxisLine . . . 106
1.6.3.3.3 Axis.CategoryAxisCrossMaxValue . . . 106
1.6.3.3.4 Axis.CategoryCrossingValue . . . 107
1.6.3.3.5 Axis.LabelFont . . . 107
1.6.3.3.6 Axis.LabelPosition . . . 108
1.6.3.3.7 Axis.LogarithmicScale . . . 109
1.6.3.3.8 Axis.MajorGridline . . . 109
1.6.3.3.9 Axis.MajorIncrement . . . 110
1.6.3.3.10 Axis.MajorTickType . . . 110
1.6.3.3.11 Axis.MaxValue . . . 111
1.6.3.3.12 Axis.MinorGridline . . . 112
1.6.3.3.13 Axis.MinorIncrement . . . 112
1.6.3.3.14 Axis.MinorTickType . . . 113
1.6.3.3.15 Axis.MinValue . . . 113
1.6.3.3.16 Axis.Number . . . 114
1.6.3.3.17 Axis.TextRotationAngle . . . 115
1.6.3.3.18 Axis.Title . . . 115
1.6.3.3.19 Axis.ValuesInReverseOrder . . . 116
1.6.3.3.20 Axis.Wall3D . . . 116
1.6.3.3.21 Axis.Wall3DBorder . . . 117
1.6.3.4 Cell . . . 117
1.6.3.4.1 Cell.Characters . . . 119
1.6.3.4.2 Cell.ClearContents() . . . 119
1.6.3.4.3 Cell.ClearFormats() . . . 119
1.6.3.4.4 Cell.Col . . . 120
1.6.3.4.5 Cell.Format . . . 120
1.6.3.4.6 Cell.Formula . . . 121
1.6.3.4.7 Cell.Name . . . 121
1.6.3.4.8 Cell.PutValue(Variant) . . . 122
1.6.3.4.9 Cell.Row . . . 122
1.6.3.4.10 Cell.SetAddInFormula(String, String) . . . 122
1.6.3.4.11 Cell.Style . . . 123
1.6.3.4.12 Cell.Value . . . 124
1.6.3.5 Cells . . . 124
1.6.3.5.1 Cells.AddHyperlink(Long, Long, Long, Long, String, String()) . . . 126
1.6.3.5.2 Cells.Cell . . . 127
1.6.3.5.3 Cells.Columns . . . 127
1.6.3.5.4 Cells.ColumnWidth . . . 128
1.6.3.5.5 Cells.CopyFromRecordset(RecordSet, Long, Long, Long, Long, Long, String, Long) . . . 128
1.6.3.5.6 Cells.GetCellAt(Long, Long) . . . 130
1.6.3.5.7 Cells.GetDataFromExcelSheet(String, String, Long, Long, Long, Long, Long) . . . 130
1.6.3.5.8 Cells.GetDataFromRecordset(RecordSet, Long, Long, Long, Long, Long, String, SAXLFieldListMode) . . . 132
1.6.3.5.9 Cells.GroupColumns(Long, Long, Boolean) . . . 133
1.6.3.5.10 Cells.GroupRows(Long, Long, Boolean) . . . 134
1.6.3.5.11 Cells.HPageBreak(SACell) . . . 135
1.6.3.5.12 Cells.Item . . . 135
1.6.3.5.13 Cells.MergeCells(Long, Long, Long, Long) . . . 136
1.6.3.5.14 Cells.PreserveStrings . . . 137
1.6.3.5.15 Cells.Range . . . 137
1.6.3.5.16 Cells.RowHeight . . . 138
1.6.3.5.17 Cells.Rows . . . 138
1.6.3.5.18 Cells.StandardHeight . . . 139
1.6.3.5.19 Cells.StandardWidth . . . 139
1.6.3.5.20 Cells.SummaryColumn . . . 139
1.6.3.5.21 Cells.SummaryRow . . . 140
1.6.3.5.22 Cells.UsedRange . . . 141
1.6.3.5.23 Cells.VPageBreak(SACell) . . . 141
1.6.3.6 Characters . . . 142
1.6.3.6.1 Characters.Font . . . 142
1.6.3.6.2 Characters.Index . . . 142
1.6.3.6.3 Characters.Length . . . 143
1.6.3.7 Chart . . . 143
1.6.3.7.1 Chart.AutoScale . . . 145
1.6.3.7.2 Chart.BopPopGapSize . . . 146
1.6.3.7.3 Chart.BopPopSecondPieSize . . . 146
1.6.3.7.4 Chart.BopPopSplitType . . . 147
1.6.3.7.5 Chart.BopPopSplitValue . . . 148
1.6.3.7.6 Chart.CategoryAxis . . . 148
1.6.3.7.7 Chart.ChartArea . . . 149
1.6.3.7.8 Chart.DataTable . . . 149
1.6.3.7.9 Chart.DataTableHasHorizontalBorders . . . 150
1.6.3.7.10 Chart.DataTableHasVerticalBorders . . . 151
1.6.3.7.11 Chart.DataTableShowsSeriesKeys . . . 151
1.6.3.7.12 Chart.FillSurfaceArea . . . 152
1.6.3.7.13 Chart.Legend . . . 152

1.6.3.7.14 Chart.LegendLocation . . . 153
1.6.3.7.15 Chart.LegendVertical . . . 154
1.6.3.7.16 Chart.Move(Long, Long, Long, Long) . . . 154
1.6.3.7.17 Chart.PlotArea . . . 155
1.6.3.7.18 Chart.RectangularCorner . . . 155
1.6.3.7.19 Chart.SecondaryCategoryAxis . . . 156
1.6.3.7.20 Chart.SecondaryValueAxis . . . 156
1.6.3.7.21 Chart.SeriesCollection . . . 157
1.6.3.7.22 Chart.SetChartType(ASXChartType, Integer) . . . 158
1.6.3.7.23 Chart.ShowDataTable . . . 158
1.6.3.7.24 Chart.ShowHorizontal . . . 159
1.6.3.7.25 Chart.ShowLegend . . . 159
1.6.3.7.26 Chart.ShowValueAsPercent . . . 160
1.6.3.7.27 Chart.SizeOfCenterHole . . . 160
1.6.3.7.28 Chart.SliceDistanceFromCenter . . . 161
1.6.3.7.29 Chart.SpaceBetweenBars . . . 161
1.6.3.7.30 Chart.SpaceBetweenCategories . . . 162
1.6.3.7.31 Chart.StackDisplayedValues . . . 162
1.6.3.7.32 Chart.Subtype . . . 163
1.6.3.7.33 Chart.Title . . . 163
1.6.3.7.34 Chart.Type . . . 164
1.6.3.7.35 Chart.Valueaxis . . . 164
1.6.3.7.36 Chart.View3D . . . 165
1.6.3.8 ChartFrame . . . 165
1.6.3.8.1 ChartFrame.Area . . . 166
1.6.3.8.2 ChartFrame.Border . . . 167
1.6.3.8.3 ChartFrame.DataLabelPlacement . . . 167
1.6.3.8.4 ChartFrame.HasShadow . . . 168
1.6.3.8.5 ChartFrame.Height . . . 168
1.6.3.8.6 ChartFrame.Number . . . 169
1.6.3.8.7 ChartFrame.Text . . . 169
1.6.3.8.8 ChartFrame.TextFont . . . 170
1.6.3.8.9 ChartFrame.TextHorizontalAlignment . . . 171
1.6.3.8.10 ChartFrame.TextRotationAngle . . . 171
1.6.3.8.11 ChartFrame.TextVerticalAlignment . . . 172
1.6.3.8.12 ChartFrame.Width . . . 172
1.6.3.8.13 ChartFrame.X . . . 173
1.6.3.8.14 ChartFrame.Y . . . 174
1.6.3.9 Charts . . . 174
1.6.3.9.1 Charts.Add(ASXChartType, Integer, Long, Long, Long, Long) . . . 175
1.6.3.9.2 Charts.Count . . . 176
1.6.3.9.3 Charts.Item . . . 176
1.6.3.10 DataLabels . . . 177
1.6.3.10.1 DataLabels.Count . . . 177
1.6.3.10.2 DataLabels.Item . . . 178
1.6.3.11 DocumentProperties . . . 178
1.6.3.11.1 DocumentProperties.Application . . . 179
1.6.3.11.2 DocumentProperties.Author . . . 180
1.6.3.11.3 DocumentProperties.Category . . . 180
1.6.3.11.4 DocumentProperties.Clear() . . . 181
1.6.3.11.5 DocumentProperties.Comments . . . 181
1.6.3.11.6 DocumentProperties.Company . . . 182
1.6.3.11.7 DocumentProperties.CreationDate . . . 182
1.6.3.11.8 DocumentProperties.Keywords . . . 183
1.6.3.11.9 DocumentProperties.LastSavedBy . . . 184
1.6.3.11.10 DocumentProperties.LastSavedDate . . . 184
1.6.3.11.11 DocumentProperties.LinksUpToDate . . . 185
1.6.3.11.12 DocumentProperties.Manager . . . 185
1.6.3.11.13 DocumentProperties.RevisionNumber . . . 186
1.6.3.11.14 DocumentProperties.ScaleCrop . . . 186
1.6.3.11.15 DocumentProperties.Subject . . . 187
1.6.3.11.16 DocumentProperties.Title . . . 187
1.6.3.12 ExcelApplication . . . 188
1.6.3.12.1 ExcelApplication.AnsiToUnicode(String, Long) . . . 189
1.6.3.12.2 ExcelApplication.ContentType . . . 190
1.6.3.12.3 ExcelApplication.CreateFont() . . . 191
1.6.3.12.4 ExcelApplication.CreateStyle() . . . 191
1.6.3.12.5 ExcelApplication.DecryptPassword . . . 192
1.6.3.12.6 ExcelApplication.DocumentProperties . . . 192
1.6.3.12.7 ExcelApplication.EncryptPassword . . . 193
1.6.3.12.8 ExcelApplication.NamedRange(String, Long) . . . 193
1.6.3.12.9 ExcelApplication.Open(Variant, Boolean) . . . 194
1.6.3.12.10 ExcelApplication.Save(String, SASaveMethod, SAFileFormat, Response) . . . 195
1.6.3.12.11 ExcelApplication.ShowHScrollBar . . . 196
1.6.3.12.12 ExcelApplication.ShowVScrollBar . . . 196

1.6.3.12.13 ExcelApplication.SpreadsheetsCreatedToday . . . 197
1.6.3.12.14 ExcelApplication.Style . . . 197
1.6.3.12.15 ExcelApplication.UnicodeToAnsi(String, Long) . . . 198
1.6.3.12.16 ExcelApplication.Version . . . 198
1.6.3.12.17 ExcelApplication.VersionEx . . . 199
1.6.3.12.18 ExcelApplication.Worksheets . . . 199
1.6.3.13 Font . . . 200
1.6.3.13.1 Font.Bold . . . 201
1.6.3.13.2 Font.Charset . . . 201
1.6.3.13.3 Font.Color . . . 201
1.6.3.13.4 Font.Italic . . . 202
1.6.3.13.5 Font.Name . . . 202
1.6.3.13.6 Font.QueryFontDisp() . . . 203
1.6.3.13.7 Font.Size . . . 203
1.6.3.13.8 Font.Strikethrough . . . 203
1.6.3.13.9 Font.Subscript . . . 204
1.6.3.13.10 Font.Superscript . . . 204
1.6.3.13.11 Font.Underline . . . 205
1.6.3.14 Line . . . 205
1.6.3.14.1 Line.LineColor . . . 206
1.6.3.14.2 Line.LineFormatting . . . 207
1.6.3.14.3 Line.LineStyle . . . 208
1.6.3.14.4 Line.LineWeight . . . 209
1.6.3.15 PageSetup . . . 209
1.6.3.15.1 PageSetup.BlackAndWhite . . . 212
1.6.3.15.2 PageSetup.BottomMargin . . . 212
1.6.3.15.3 PageSetup.CenterFooter . . . 213
1.6.3.15.4 PageSetup.CenterHeader . . . 213
1.6.3.15.5 PageSetup.CenterHorizontally . . . 214
1.6.3.15.6 PageSetup.CenterVertically . . . 214
1.6.3.15.7 PageSetup.Draft . . . 215
1.6.3.15.8 PageSetup.FirstPageNumber . . . 215
1.6.3.15.9 PageSetup.FitToPagesTall . . . 216
1.6.3.15.10 PageSetup.FitToPagesWide . . . 216
1.6.3.15.11 PageSetup.FooterMargin . . . 217
1.6.3.15.12 PageSetup.GetPrintTitles() . . . 217
1.6.3.15.13 PageSetup.HeaderMargin . . . 218
1.6.3.15.14 PageSetup.LeftFooter . . . 218
1.6.3.15.15 PageSetup.LeftHeader . . . 219
1.6.3.15.16 PageSetup.LeftMargin . . . 219
1.6.3.15.17 PageSetup.Order . . . 220
1.6.3.15.18 PageSetup.Orientation . . . 220
1.6.3.15.19 PageSetup.PaperSize . . . 221
1.6.3.15.20 PageSetup.PrintArea . . . 223
1.6.3.15.21 PageSetup.PrintComments . . . 223
1.6.3.15.22 PageSetup.PrintGridlines . . . 224
1.6.3.15.23 PageSetup.PrintHeadings . . . 224
1.6.3.15.24 PageSetup.PrintNotes . . . 225
1.6.3.15.25 PageSetup.RightFooter . . . 225
1.6.3.15.26 PageSetup.RightHeader . . . 226
1.6.3.15.27 PageSetup.RightMargin . . . 226
1.6.3.15.28 PageSetup.SetPrintTitleColumns(Long, Long) . . . 227
1.6.3.15.29 PageSetup.SetPrintTitleRows(Long, Long) . . . 227
1.6.3.15.30 PageSetup.TopMargin . . . 228
1.6.3.15.31 PageSetup.UseZoom . . . 228
1.6.3.15.32 PageSetup.Zoom . . . 229
1.6.3.16 Picture . . . 229
1.6.3.16.1 Picture.AreaPattern . . . 230
1.6.3.16.2 Picture.BorderLineType . . . 231
1.6.3.16.3 Picture.BorderWeight . . . 232
1.6.3.16.4 Picture.HasShadow . . . 233
1.6.3.16.5 Picture.LowerDeltaX . . . 233
1.6.3.16.6 Picture.LowerDeltaY . . . 234
1.6.3.16.7 Picture.LowerRightColumn . . . 234
1.6.3.16.8 Picture.LowerRightRow . . . 235
1.6.3.16.9 Picture.Move(Long, Long, Long, Long) . . . 235
1.6.3.16.10 Picture.UpperDeltaX . . . 236
1.6.3.16.11 Picture.UpperDeltaY . . . 236
1.6.3.16.12 Picture.UpperLeftColumn . . . 237
1.6.3.16.13 Picture.UpperLeftRow . . . 237
1.6.3.17 Pictures . . . 238
1.6.3.17.1 Pictures.Count . . . 238
1.6.3.17.2 Pictures.CreateFromFile2(Long, Long, String) . . . 239
1.6.3.17.3 Pictures.CreateFromFile(Long, Long, Long, Long, String) . . . 240
1.6.3.17.4 Pictures.Item . . . 241

1.6.3.18 PivotField . . . 241
1.6.3.18.1 PivotField.AddSubtotal(SAXlSubtotalType) . . . 243
1.6.3.18.2 PivotField.AutoShowCount . . . 244
1.6.3.18.3 PivotField.AutoShowField . . . 244
1.6.3.18.4 PivotField.AutoShowRange . . . 244
1.6.3.18.5 PivotField.AutoShowType . . . 245
1.6.3.18.6 PivotField.AutoSortField . . . 245
1.6.3.18.7 PivotField.AutoSortType . . . 246
1.6.3.18.8 PivotField.ClearSubtotals() . . . 246
1.6.3.18.9 PivotField.DataDisplayFormat . . . 247
1.6.3.18.10 PivotField.DisplayName . . . 248
1.6.3.18.11 PivotField.InOutlineForm . . . 248
1.6.3.18.12 PivotField.IsCalculatedDataField . . . 248
1.6.3.18.13 PivotField.Name . . . 249
1.6.3.18.14 PivotField.NumberFormat . . . 249
1.6.3.18.15 PivotField.RemoveSubtotal(SAXlSubtotalType) . . . 249
1.6.3.18.16 PivotField.SummarizeBy . . . 250
1.6.3.18.17 PivotField.Type . . . 251
1.6.3.19 PivotFields . . . 251
1.6.3.19.1 PivotFields.Count . . . 252
1.6.3.19.2 PivotFields.Item . . . 252
1.6.3.20 PivotTable . . . 252
1.6.3.20.1 PivotTable.AddDataField(IPivotField, SAXlSummarizationType) . . . 254
1.6.3.20.2 PivotTable.AutoFormatType . . . 256
1.6.3.20.3 PivotTable.DataCollectionInRow . . . 257
1.6.3.20.4 PivotTable.EmptyCellsValue . . . 258
1.6.3.20.5 PivotTable.EnableDrillDown . . . 258
1.6.3.20.6 PivotTable.EnableFieldDialog . . . 259
1.6.3.20.7 PivotTable.EnableWizard . . . 259
1.6.3.20.8 PivotTable.Fields . . . 259
1.6.3.20.9 PivotTable.FieldsPerPage . . . 260
1.6.3.20.10 PivotTable.FriendlyErrorValue . . . 260
1.6.3.20.11 PivotTable.GrandTotalsForColumns . . . 261
1.6.3.20.12 PivotTable.GrandTotalsForRows . . . 261
1.6.3.20.13 PivotTable.InOutlineForm . . . 261
1.6.3.20.14 PivotTable.MergeLabels . . . 262
1.6.3.20.15 PivotTable.Move(Long, Long) . . . 262
1.6.3.20.16 PivotTable.Name . . . 262
1.6.3.20.17 PivotTable.PageLayout . . . 263
1.6.3.20.18 PivotTable.PreserveFormatting . . . 263
1.6.3.20.19 PivotTable.RepeatItemLabels . . . 264
1.6.3.20.20 PivotTable.SetPrintTitles . . . 264
1.6.3.20.21 PivotTable.ShowFriendlyErrorValue . . . 265
1.6.3.20.22 PivotTable.ShowValueForEmptyCells . . . 265
1.6.3.20.23 PivotTable.SubtotalHiddenPageItems . . . 265
1.6.3.20.24 PivotTable.UpperLeftColumn . . . 266
1.6.3.20.25 PivotTable.UpperLeftRow . . . 266
1.6.3.21 PivotTables . . . 266
1.6.3.21.1 PivotTables.Count . . . 267
1.6.3.21.2 PivotTables.CreatePivotTable(SARange, Long, Long) . . . 267
1.6.3.21.3 PivotTables.Item . . . 268
1.6.3.22 Range . . . 269
1.6.3.22.1 Range.AddHyperlink(String, String) . . . 270
1.6.3.22.2 Range.Area . . . 271
1.6.3.22.3 Range.AreaCount . . . 272
1.6.3.22.4 Range.AutoFitHeight() . . . 272
1.6.3.22.5 Range.AutoFitWidth() . . . 272
1.6.3.22.6 Range.BorderAround(SABorderLineStyle, Long) . . . 273
1.6.3.22.7 Range.Cell . . . 274
1.6.3.22.8 Range.ColumnCount . . . 274
1.6.3.22.9 Range.ColumnWidth . . . 275
1.6.3.22.10 Range.FirstColumn . . . 275
1.6.3.22.11 Range.FirstRow . . . 276
1.6.3.22.12 Range.GetCellAt(Integer, Integer) . . . 276
1.6.3.22.13 Range.GroupColumns(Long, Long, Boolean) . . . 277
1.6.3.22.14 Range.GroupRows(Long, Long, Boolean) . . . 277
1.6.3.22.15 Range.Intersect(SARange) . . . 278
1.6.3.22.16 Range.Item . . . 279
1.6.3.22.17 Range.MergeCells() . . . 279
1.6.3.22.18 Range.Name . . . 280
1.6.3.22.19 Range.RowCount . . . 280
1.6.3.22.20 Range.RowHeight . . . 281
1.6.3.22.21 Range.Style . . . 281
1.6.3.22.22 Range.Union(SARange) . . . 282
1.6.3.22.23 Range.UsedRange . . . 282

1.6.3.22.24 Range.Value . . . 283
1.6.3.23 Series . . . 283
1.6.3.23.1 Series.Area . . . 285
1.6.3.23.2 Series.AxisGroup . . . 286
1.6.3.23.3 Series.Border . . . 287
1.6.3.23.4 Series.ChartSubType . . . 287
1.6.3.23.5 Series.ChartType . . . 288
1.6.3.23.6 Series.DataLabels . . . 289
1.6.3.23.7 Series.ErrorBar . . . 289
1.6.3.23.8 Series.ErrorBarLine . . . 290
1.6.3.23.9 Series.HasSmoothedLine . . . 291
1.6.3.23.10 Series.LabelFrame . . . 291
1.6.3.23.11 Series.LabelShowActualValue . . . 292
1.6.3.23.12 Series.LabelShowCategoryAndValueAsPercent . . . 292
1.6.3.23.13 Series.LabelShowCategoryLabel . . . 293
1.6.3.23.14 Series.LabelShowValueAsPercent . . . 294
1.6.3.23.15 Series.MarkerFormat . . . 294
1.6.3.23.16 Series.MarkerSize . . . 295
1.6.3.23.17 Series.MarkerType . . . 295
1.6.3.23.18 Series.Name . . . 296
1.6.3.23.19 Series.ScatterValues . . . 297
1.6.3.23.20 Series.ShowInLegend . . . 297
1.6.3.23.21 Series.SpaceBetweenBars . . . 298
1.6.3.23.22 Series.SpaceBetweenCategories . . . 298
1.6.3.23.23 Series.Values . . . 299
1.6.3.23.24 Series.XErrorBarCustomMinusValues . . . 299
1.6.3.23.25 Series.XErrorBarCustomPlusValues . . . 300
1.6.3.23.26 Series.XErrorBarTShaped . . . 300
1.6.3.23.27 Series.XErrorBarValue . . . 301
1.6.3.23.28 Series.XErrorBarValueSource . . . 302
1.6.3.23.29 Series.YErrorBarCustomMinusValues . . . 303
1.6.3.23.30 Series.YErrorBarCustomPlusValues . . . 303
1.6.3.23.31 Series.YErrorBarTShaped . . . 304
1.6.3.23.32 Series.YErrorBarValue . . . 305
1.6.3.23.33 Series.YErrorBarValueSource . . . 305
1.6.3.24 SeriesCollection . . . 306
1.6.3.24.1 SeriesCollection.Add(Area, , Long) . . . 307
1.6.3.24.2 SeriesCollection.CategoryData . . . 308
1.6.3.24.3 SeriesCollection.Item . . . 308
1.6.3.24.4 SeriesCollection.LabelShowActualValue . . . 309
1.6.3.24.5 SeriesCollection.LabelShowCategoryAndValueAsPercent . . . 309
1.6.3.24.6 SeriesCollection.LabelShowCategoryLabel . . . 310
1.6.3.24.7 SeriesCollection.LabelShowValueAsPercent . . . 310
1.6.3.25 Style . . . 311
1.6.3.25.1 Style.BackgroundColor . . . 312
1.6.3.25.2 Style.BorderColor . . . 313
1.6.3.25.3 Style.BorderLineStyle . . . 314
1.6.3.25.4 Style.Font . . . 315
1.6.3.25.5 Style.ForegroundColor . . . 315
1.6.3.25.6 Style.Hidden . . . 316
1.6.3.25.7 Style.HorizontalAlignment . . . 316
1.6.3.25.8 Style.Indent . . . 317
1.6.3.25.9 Style.Locked . . . 317
1.6.3.25.10 Style.Name . . . 318
1.6.3.25.11 Style.Number . . . 318
1.6.3.25.12 Style.Orientation . . . 318
1.6.3.25.13 Style.Pattern . . . 319
1.6.3.25.14 Style.ShrinkToFit . . . 320
1.6.3.25.15 Style.VerticalAlignment . . . 320
1.6.3.25.16 Style.WrapText . . . 321
1.6.3.26 Worksheet . . . 321
1.6.3.26.1 Worksheet.Cells . . . 323
1.6.3.26.2 Worksheet.Charts . . . 323
1.6.3.26.3 Worksheet.Copy() . . . 323
1.6.3.26.4 Worksheet.FreezePanes(SACell) . . . 324
1.6.3.26.5 Worksheet.Name . . . 324
1.6.3.26.6 Worksheet.PageSetup . . . 325
1.6.3.26.7 Worksheet.Pictures . . . 325
1.6.3.26.8 Worksheet.PivotTables . . . 326
1.6.3.26.9 Worksheet.SelectSheet() . . . 326
1.6.3.26.10 Worksheet.ShowGridlines . . . 327
1.6.3.26.11 Worksheet.ShowRowColHeaders . . . 327
1.6.3.26.12 Worksheet.Visible . . . 327
1.6.3.26.13 Worksheet.WriteProtect . . . 328
1.6.3.26.14 Worksheet.Zoom . . . 329

1.6.3.27 Worksheets . . . 329
1.6.3.27.1 Worksheets.Add(String) . . . 330
1.6.3.27.2 Worksheets.Count . . . 330
1.6.3.27.3 Worksheets.Item . . . 331
1.6.3.27.4 Worksheets.Remove(Variant) . . . 331
1.6.4 Formula Functions . . . 332
1.6.4.1 Database Functions . . . 334
1.6.4.2 Financial Functions . . . 338
1.6.4.3 Information Functions . . . 339
1.6.4.4 Logical Functions . . . 340
1.6.4.5 Lookup and Reference Functions . . . 341
1.6.4.6 Math Functions . . . 342
1.6.4.7 Statistical Functions . . . 347
1.6.4.8 String Functions . . . 349
1.6.4.9 Time Functions . . . 350
1.6.5 Formula Calculation Operators . . . 353
1.6.6 Formatting Codes . . . 354
1.6.7 Chart Codes . . . 356
1.6.8 PivotTable AutoFormat Types . . . 369
1.7 Installation . . . 378
1.7.1 Installing ExcelWriter . . . 378
1.7.2 Removing ExcelWriter . . . 379
1.7.3 Configuring IIS . . . 380
1.7.4 LockedFileRetry Parameter . . . 381
1.7.5 Security Considerations . . . 381
1.8 OfficeWriter Links . . . 381
1.9 External Links . . . 381

ExcelWriter COM 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.
- **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.
- **Installation**
  includes instructions for automatic and manual installation. System Administrators will find useful information here.
- **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

- The Web Reporting Solution
- What is ExcelWriter?
- Features and Benefits
- ExcelWriter Editions
- Requirements
- Frequently Asked Questions
- Change Log

The Web Reporting Solution

SoftArtisans ExcelWriter: The Power of Microsoft® Excel on the Web

SoftArtisans ExcelWriter is a powerful and scalable solution for producing online query and reporting applications in the format that end-users prefer: Microsoft Excel. A server-side product, ExcelWriter produces richly formatted, native Excel workbooks on-the-fly for an unlimited amount of concurrent end-users, without Microsoft Excel on the server. With SoftArtisans ExcelWriter, online applications and intranet portals seamlessly integrate with common desktop applications.

With SoftArtisans ExcelWriter, standard Excel documents do not have to be compromised into HTML, flat CSV files, or uneditable PDF files to be quickly served through the Web. ExcelWriter produces real, native Excel – not Excel lookalikes. A simple yet comprehensive solution, ExcelWriter can use dynamic data from a database or an existing spreadsheet. ExcelWriter enables a wide range of businesses to create high-performance applications while making efficient use of limited systems, development resources, and users' Excel proficiency.

Benefits

The benefits of using SoftArtisans ExcelWriter for online reporting include:

- **Familiarity of Client**
  Microsoft Excel is the most widely used spreadsheet application currently available to work with quantitative data. The widespread familiarity with Excel eliminates the need for comprehensive training on a proprietary Web reporting software package.

- **Richness**
  ExcelWriter delivers richly formatted native Excel files that contain comprehensive formatting, formulas, custom styles, charting, images, and the ability to precisely control page layout, set page breaks, and format print options.

- **Ease of Modification**
  Once received on the client machine, the spreadsheets can be manipulated, sorted, and modified as required. Alternatively, these Excel files can be secured and protected, client-side, to avoid tampering or unauthorized access.

- **Upload Back to the Server**
  Excel files that have been modified client-side, can be uploaded back to the Web server and read with Cells.GetDataFromExcelSheet, or modified with ExcelApplication.Open.

- **Flexibility**
  Users can serve themselves data and generate Excel reports from any location that has Internet access.

- **Delivery Speed**
  Because Excel is not required on the server, an unlimited number of reports can be generated without straining the Web server.

- **Productivity**
  Business administration can become more automated. Custom reports for different departments/regions can be automated, reducing dependency on the IT department.

- **Data Visualization and Analysis**
  Native Excel charting (2D & 3D) makes data easy to understand.

- **Consistency**
  Same data and formatting in reports across the corporation - frequently accessed reports can be saved on the server.

- **Ease of Use**
  ExcelWriter makes Web reports accessible to all end users, not just specially trained power users. Most business people are familiar and proficient with Microsoft Excel as compared to other Web reporting tools.

- **Time Management**
  Reporting via the Web reduces and eliminates meetings, conference calls, and many customer service issues.

Web Reporting Format Comparison

	Microsoft Excel	Adobe® PDF	HTML	CSV
Modifiable by users	Yes	No	No	Yes
Multi-platform support	Yes	Yes	Yes	Yes
Used for Business Intelligence/Decision Support	Yes	No	No	Moderately
Image support	Yes	Yes	Yes	No
Dynamic chart support	Yes	No	No	No
Upload back to server	Yes	No	No	Yes
Can open in browser	Yes	Yes	Yes	Yes
Rich formatting	Yes	Yes	Moderately	No

What is ExcelWriter?

SoftArtisans ExcelWriter is a high-performance 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) 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, an ExcelWriter template, or generate an Excel file 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.
Pure Coding	**Completely Dynamic** This dynamic approach provides total control, at runtime, of every element in the resulting spreadsheet.

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 generates native Excel files via server-side scripting that can be read by Microsoft Excel.

ExcelWriter does not use XML or HTML to represent the spreadsheet. The resultant spreadsheet is in the native Microsoft binary format (BIFF8). This permits the creation of rich spreadsheets, including charts and images, and allows for support of all versions of Excel from Excel 95 through Excel XP.

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. (Please note: 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

- Generates full function, native Microsoft Excel from a Web form, an existing Excel workbook, or any data source.
- Built for the most demanding sites, ExcelWriter easily generates thousands of workbooks per day.
- Code-based or template-driven development.
- Server-side solution: does not require Microsoft Excel on the user's computer or the server.
- Enables easy spreadsheet generation from a database.
- HotCell Technology allows you to update a server-side data source directly from client-side Microsoft Excel.
- Produces editable Excel spreadsheets (BIFF8 or BIFF7).
- Supports virtually all Excel formatting, functions, and charts.
- Supports advanced Excel features such as pivot tables, VBA, and macros.
- Creates pictures from existing image files.
- Allows you to save the generated workbook to disk or stream it to a Web browser.
- ExcelWriter is a high performance, multithreaded, single dynamic link library (DLL).
- Reads existing Excel spreadsheets and allows you to modify existing spreadsheets.
- Can be used from ASP, ASP.NET, or Visual Basic.

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** is for high-volume sites that require less Excel formatting, functions, and charting at runtime. There is no limit to the number of spreadsheets ExcelWriterSE can generate.

- **OfficeWriter**
  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 typical 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	✔	✔
Pure Code-based Development	✔	---
Reporting Services Integration	✔	Limited
OfficeWriter Assistant (file-transfer tool)	✔	---
Spreadsheets Delivered Per Day	Unlimited	Unlimited

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, though ExcelWriter COM must be run in 32-bit mode.
- IIS 5 or higher.
- ASP 5.0 or higher.

If you intend to create spreadsheets from databases, install the latest Microsoft drivers. The driver package is available for free for all versions of IIS at http://www.microsoft.com/data. Choose to download MDAC 2.0 or later. MDAC includes updates to ADO, OLEDB, and related ODBC drivers.

Client Requirements

No software is required on the client to download and save an Excel Spreadsheet. To open Excel spreadsheets, one of the following must be installed on the client:

- Microsoft Excel 95, 97, 2000, 2002 (XP), 2003, or 2007.
- 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 spreadsheet application that supports the Excel format.
- To enable full functionality of HotCell Technology, the OfficeWriter Assistant cab file (OWAssist.cab) must be downloaded and installed on the client. ExcelWriter Enterprise Edition includes OfficeWriter Assistant. OfficeWriter Assistant is not included with ExcelWriterSE.

Frequently Asked Questions

1. Does ExcelWriter read existing spreadsheets?
2. Can ExcelWriter modify an existing spreadsheet?
3. Can I generate a chart using ExcelWriter?
4. What versions of Microsoft Excel does ExcelWriter support?
5. Which Microsoft Excel features are not available in ExcelWriter?
6. Does ExcelWriter include features that are not available in Microsoft Excel?
7. What software do I have to have on my server to use ExcelWriter?
8. Can I include pictures in ExcelWriter generated spreadsheets?

1. **Does ExcelWriter read existing spreadsheets?**

   Yes. To read data from an existing spreadsheet, use the method GetDataFromExcelSheet, or open the file using ExcelApplication.Open.

2. **Can ExcelWriter modify an existing spreadsheet?**

   Yes. See Modifying an Existing Spreadsheet.

3. **Can I generate a chart using ExcelWriter?**

   Yes. Use the method Charts.Add to generate a chart. ExcelWriter can generate 15 chart types and over 70 sub-types (see Chart Codes). ExcelWriter's numerous charting properties allow you to customize every detail of your chart.

4. **What versions of Microsoft Excel does ExcelWriter support?**

   ExcelWriter can produce either Excel 97/2000/2002/2003 (BIFF8) format files or Excel 95 (BIFF7) format files. Not all features are supported by Excel 95.

   > The ExcelTemplate object supports Excel's BIFF8 (Excel 97/2000/2002/2003) format only; ExcelTemplate does not support BIFF7 (Excel 95) format templates, and will not generate BIFF7 format files.

5. **Which Microsoft Excel features are not available in ExcelWriter?**

   We are constantly adding new features to ExcelWriter. If there is a particular Microsoft Excel feature currently unavailable in ExcelWriter that you would like to use, please let us know by sending an email to info@softartisans.com.

   - The ExcelApplication and ExcelTemplate objects cannot create certain Excel features such as Pivot Tables, Macros and VBA (Visual Basic for Applications). However, ExcelWriter will preserve all Excel features when opening an existing spreadsheet. Creating a file in Microsoft Excel and using it as input to ExcelWriter allows you to use any Excel feature. ExcelWriter will apply changes in memory to the XLS file, such as by merging against a database or changing formatting. The modified file can be streamed to the user's browser, saved as a new file, or saved over the original file. See Modifying an Existing Spreadsheet and How to Use Templates.
   - ExcelWriter cannot create a hyperlink to a different workbook.

6. **Does ExcelWriter include features that are not available in Microsoft Excel?**

   Yes, ExcelWriter includes the following features unavailable in Microsoft Excel:

   - Single step import from a database. See Importing Data from a Database.
   - Ability to stream spreadsheet back to the browser. See Output Options.
   - Template based data binding. See How to Use Templates.
   - The ability to update a server-side data source directly from client-side Microsoft Excel. See HotCell Technology

7. **What software do I have to have on my server to use ExcelWriter?**

   - Windows NT 4, Windows 2000, Windows XP, or Windows 2003 Server.
   - IIS 4 or higher.
   - ASP 5.0 or higher.

8. **Can I include pictures in ExcelWriter generated spreadsheets?**

   Yes. To add a picture to your spreadsheet, use Pictures.CreateFromFile(Long, Long, Long, Long, String).

Change Log

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

Differences between 7.5.1 and 7.6

- ExcelWriter COM is no longer installed by the OfficeWriter automatic installer. If you would like to install ExcelWriter COM, please contact SoftArtisans.

Differences between 7.1 and 7.5.1

Issue resolved

- License Key Manager threw error when being run on a machine with no SoftArtisans license keys in the registry.

Differences between 7.0 and 7.1

Issue resolved

- ExcelApplication COM threw an unknown exception when opening a large template under heavy load

Quick Start

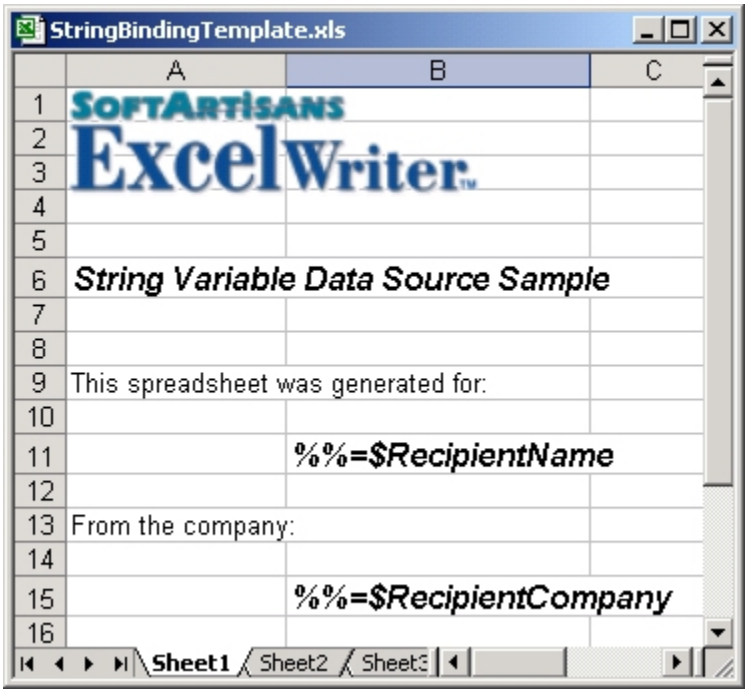
- Create a Spreadsheet with ExcelTemplate
- Create a Spreadsheet with ExcelApplication

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, and %%=$ specifies a variable or array.

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.



A template spreadsheet with two string variable data markers.

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: %%=$RecipientNameand %%=$RecipientCompany. %%=$ (rather than %%=) indicates that the data source is a variable or an array. In code, ExcelWriter sets the data sources for the data markers to two simple string variables.

Step 1: Create a Template

1. In Microsoft Excel, create a new file.
2. In one or more cells, enter data markers. For example, in cell B11 enter the variable data marker %%=$RecipientName and in cell B15 enter the variable data marker %%=$RecipientCompany.
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 script that uses ExcelTemplate to open the template, populate data markers, and generate a new Excel spreadsheet. The following lines of code generate a new spreadsheet from a template that contains two data markers: %%=$RecipientNameand %%=$RecipientCompany. We will set the data sources for the data markers to two simple string variables.

The ExcelTemplate object represents the template Excel file. To generate a new spreadsheet using ExcelTemplate:

1. Add the following METADATA directive to the ASP script to access ExcelWriter constants:

   ```
   <!--METADATA TYPE="TypeLib" UUID="{7BCD2133-64A0-4770-843C-090637114583}" -->
   ```

   In ASP, Typelibs provide quick and convenient access to constants associated with a particular object. The script in this example includes ExcelWriter's TypeLib because the script uses the constant **saProcessOpenInExcel**. The UUID attribute specifies ExcelWriter's unique identifier.

2. Create an instance of the ExcelTemplate object, for example:

   ```
   Set XlwTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")}
   ```

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

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

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

4. Use the ExcelTemplate.DataSource property to assign data sources to bind to the template's data markers, for example:

   ```
   '--- Bind the variables to the template data markers
   '--- %%=$RecipientName and %%=$RecipientCompany
   XlwTemplate.DataSource("RecipientName") = "J. Smith"
   XlwTemplate.DataSource("RecipientCompany") = "SoftArtisans"
   ```

5. Call ExcelTemplate.Process to populate the template's data markers with data source values and generate a new spreadsheet:

   ```
   XlwTemplate.Process "StringBinding.xls", saProcessOpenInExcel, True
   ```

   The Process method's second parameter specifies whether to save the generated spreadsheet on the server, send to the browser and open it in Excel, send to the browser and open it in the browser window, or return an ExcelApplication object. If the third parameter is set to true (as in the example), macros in the template will be excluded from the generated spreadsheet; to preserve macros, set the parameter to false.

Create a Spreadsheet with ExcelApplication

To create generate an Excel workbook from code without using an ExcelWriter template, use ExcelApplication, the main class for pure code-based workbook generation. An ExcelApplication object represents an Excel workbook, and is also an engine used to open, create, and write (save or stream to a browser) the workbook.

To generate a workbook with ExcelApplication:

1. Create an instance of ExcelApplication.

```
Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
```

2. Create a worksheet:

```
Set ws = xlw.Worksheets(1)
```

   A spreadsheet may contain multiple worksheets. The first worksheet in a spreadsheet is WorkSheets(1). If a spreadsheet contains more than one worksheet, the second would be WorkSheets(2), the third WorkSheets(3), and so on. This example creates only one worksheet.

3. Within the new worksheet, create cells and fill in values:

```
ws.Cells("A1").Value = "Name"
ws.Cells("B1").Value = "Count"
ws.Cells("C1").Value = "Dollar"

ws.Cells("A2").Value = "Fred Smith"
ws.Cells("B2").Value = 10
ws.Cells("C2").Value = 37.5
```

4. When the spreadsheet is complete, you can save it to the server's hard disk, return it in memory, return it as an ExcelWriter template, stream it to the browser. In this example, ExcelWriter streams the spreadsheet to the browser:

```
xlw.Save "first.xls", saOpenInPlace
```

5. Explicitly destroy the ExcelApplication object:

```
Set xlw = Nothing
Response.End
```

   Destroying the ExcelApplication object is not strictly necessary but is considered good programming practice. The HTTP response is also ended. This prevents extraneous additional characters from being sent along with the spreadsheet.

The spreadsheet was in memory and is now sent to the user's browser. The user may choose to save the spreadsheet or open it. If it is saved, the default name will be **first.xls**. If the user opens the file, it will open in the browser because the second parameter passed to Save is saOpenInPlace.

Note that the code does not contain HTML. This is because the generated spreadsheet is written (to disk, memory, or the client) in a single HTTP response. If the response includes HTML, the spreadsheet will be corrupted. The HTTP response should contain only the creation of the spreadsheet. **Do not include HTML or Response.Write lines.**

Avoiding Errors by Using Typelibs

In ASP, Typelibs provide quick and convenient access to constants associated with a particular object. The script in this example includes ExcelWriter's TypeLib because the script uses the constant **saOpenInPlace**. Avoid errors by always including the TypeLib in your ExcelWriter scripts. The UUID attribute specifies ExcelWriter's unique identifier:

```
<!--METADATA TYPE="TypeLib" UUID="{7BCD2133-64A0-4770-843C-090637114583}" -->
```

Section Summary

1. The ExcelWriter script should only contain the server-side code that generates the spreadsheet. Any HTML or Response.Write commands will corrupt the spreadsheet.

2. The first step is to create an instance of the ExcelApplication object.

3. A spreadsheet contains multiple **worksheets**, which in turn contain multiple **cells**.

4. The spreadsheet can be streamed to the browser in one step. It does not have to be stored on the Web server's hard disk.

ExcelTemplate in Depth

- How to Use Templates
- Creating Data Markers
- Using an Array as a Data Source
- Using a Database as a Data Source
- Output Options
- Templates and Charts
- Templates and PivotTables
- Passing ExcelTemplate to ExcelApplication

How to Use Templates

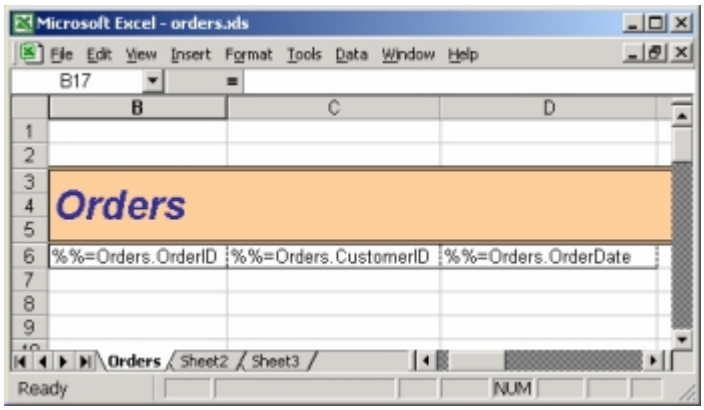
An ExcelWriter **template** is an Excel spreadsheet that contains ExcelWriter data markers. A data marker is a cell value beginning with **%%=** or **%%=$** that specifies a database column, variable, or array to insert in the spreadsheet column containing the marker. A data marker may include modifiers.

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 a data set created from the Orders table in a database, ExcelWriter will import the OrderID column to the spreadsheet's column B. In addition to data markers, your template workbook can include any values, formulas, formatting, charts, pivot tables, etc.

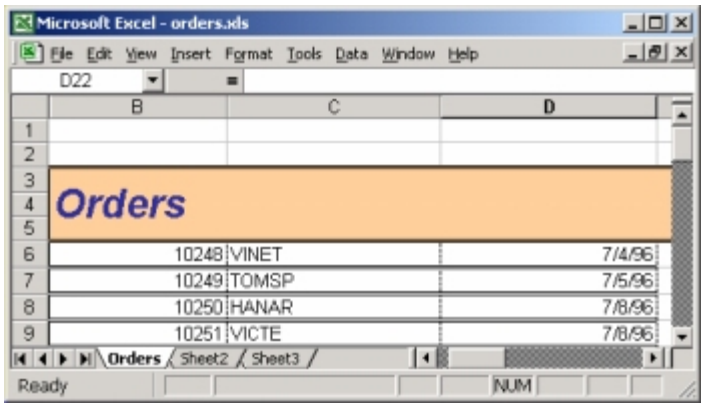
A data marker binds in script to a **data source** which may be the result of a database query, a variable, or an array, and may include modifiers. Data source and field numbers are 1-based. If ExcelWriter encounters **%%=#0[.field]** or **%%=[DataSource.]#0**, an error will occur.

> 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.

Template



Spreadsheet generated from template



> A template should not include strings that contain more than one font. These strings will not be preserved in the generated spreadsheet.

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

1. Create an instance of the ExcelTemplate object, for example:

```
Set XlwTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")
```

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

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

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

3. Use the DataSource property to assign data sources to bind to the template's data markers, for example:

```
'--- Bind the variables to the template data markers
'--- %%=$RecipientName and %%=$RecipientCompany
XlwTemplate.DataSource("RecipientName") = "J. Smith"
XlwTemplate.DataSource("RecipientCompany") = "SoftArtisans"
```

4. Call Process to populate the template's data markers with data source values and generate a new spreadsheet:

```
XlwTemplate.Process "StringBinding.xls", saProcessOpenInExcel, True
```

   The Process method's second parameter specifies whether to save the generated spreadsheet on the server, send it to the browser and open it in Excel, send to the browser and open it in the browser window, or return an ExcelApplication object. If the third parameter is set to true (as in the example), macros in the template will be excluded from the generated spreadsheet; to preserve macros, set the parameter to false.

Creating Data Markers

- Data Markers
- Data Marker Names
- Data Marker Formats
- Using Short Data Markers
- Data Marker Modifiers

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 **%%=** 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.

A data marker binds in script to a **data source** which may be a variable, an array, or an ADO recordset. 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.

The template ArrayBindingTemplate.xls contains several data markers. The following table shows you how to bind the data markers to array data sources in code.

In this sample, the data markers are bound to data sources in code as follows:

Data Marker	Code

%%=$SimpleArray	`Dim SimpleArray(6)`
	`SimpleArray(0) = "Sunday"`
	`SimpleArray(1) = "Monday"`
	`SimpleArray(2) = "Tuesday"`
	`SimpleArray(3) = "Wednesday"`
	`SimpleArray(4) = "Thursday"`
	`SimpleArray(5) = "Friday"`
	`SimpleArray(6) = "Saturday"`
	`XlwTemplate.DataSource("SimpleArray") = SimpleArray`

%%=$TwoDimArray.#1
%%=$TwoDimArray.#2
%%=$TwoDimArray.#3
	`Dim TwoDimArray(1, 2)`
	`TwoDimArray(0, 0) = "Nancy Davolio"`
	`TwoDimArray(0, 1) = "Sales Manager"`
	`TwoDimArray(0, 2) = "Northwind Sales Dept."`
	`TwoDimArray(1, 0) = "Andrew Suyama"`
	`TwoDimArray(1, 1) = "Analyst"`
	`TwoDimArray(1, 2) = "Northwind Finance Dept."`
	
	`XlwTemplate.DataSource("TwoDimArray") = TwoDimArray`

%%=$TwoDimArray2.#1
 	
%%=$TwoDimArray2.#2
 	
%%=$TwoDimArray2.#3
	`Dim TwoDimArray2(0, 2)`
	`TwoDimArray2(0, 0) = "2005"`
	`TwoDimArray2(0, 1) = "2006"`
	`TwoDimArray2(0, 2) = "2007"`
	`XlwTemplate.DataSource("TwoDimArray2") =`
	`TwoDimArray2`

Data Marker Names

Follow these rules when naming data markers:

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

To include spaces 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 are:
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890_ @$#

Data Marker Formats

Data Marker Formats	
For this data source:	**Use one of these data marker formats:**
Recordset	**%%=DataSourceName.ColumnName[(modifier)]** **%%=#DataSourceNumber.#ColumnNumber[(modifier)]** **%%=#DataSourceNumber.ColumnName[(modifier)]** **%%=DataSourceName.#ColumnNumber[(modifier)]** **%%=ColumnName[(modifier)]** **%%=#ColumnNumber[(modifier)]**

Variable	**%%=$DataSourceName[(modifier)]** **%%=$#DataSourceNumber[(modifier)]**
One-Dimensional Array data marker	**%%=$DataSourceName[(modifier)]** **%%=$#DataSourceNumber[(modifier)]**
Two-dimensional Array data marker	**%%=$DataSourceName.#ColumnNumber[(modifier)]** **%%=$#DataSourceNumber.#ColumnNumber[(modifier)]** **%%=$#ColumnNumber[(modifier)]**

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 $.

> A data marker that binds to a variable, a 1-d array, or a 2-d array must begin with %%=$ (not %%=), whether the data marker format is short or long.

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.

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.
%%=$DataSourceName	**%%=$OrdersFirst** column in the data source "Orders."

Data Marker Modifiers

An ExcelWriter data marker can include modifiers. The format for a data marker with a modifier is **%%=datamarker(*modifier*)**. 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 **%%=Recordset.#2(fieldname)**, ExcelWriter will insert the name of recordset 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 in the template spreadsheet 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 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. In ASP, If a data marker contains a column number (for example **%%=Orders.#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.

Using an Array as a Data Source

The following example generates an Excel spreadsheet from the template ArrayBindingTemplate.xls. The template contains several data markers that bind to array data sources in the ASP code.

```
'--- Declarations
Dim SimpleArray(6) '--- 1-D array
Dim TwoDimArray(2, 2) '--- 2-D array, 3 rows
Dim TwoDimArray2(0, 2) '--- 2-D array, 1 row
Dim XlwTemplate

'--- Populate a one dimensional array with values.
SimpleArray(0) = "Sunday"
SimpleArray(1) = "Monday"
SimpleArray(2) = "Tuesday"
SimpleArray(3) = "Wednesday"
SimpleArray(4) = "Thursday"
SimpleArray(5) = "Friday"
SimpleArray(6) = "Saturday"

'--- Populate a two-dimensional array with values.
TwoDimArray(0, 0) = "Nancy Davolio"
TwoDimArray(0, 1) = "Sales Manager"
TwoDimArray(0, 2) = "Northwind Sales Dept."
TwoDimArray(1, 0) = "Andrew Suyama"
TwoDimArray(1, 1) = "Analyst"
TwoDimArray(1, 2) = "Northwind Finance Dept."
TwoDimArray(2, 0) = "Yorba Von Schnitzel"
TwoDimArray(2, 1) = "Manager"
TwoDimArray(2, 2) = "IS Support"

'--- Create a two-dimensional array with one row.
TwoDimArray2(0, 0) = "2005"
TwoDimArray2(0, 1) = "2006"
TwoDimArray2(0, 2) = "2007"

'--- Create an instance of ExcelTemplate.
Set XlwTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")

'--- Open the template workbook
XlwTemplate.Open(C:\Templates\ArrayBindingTemplateASP.xls")
CheckError Err, "There was an error opening the template workbook."

'--- Bind SimpleArray to the data marker
'--- %%=$SimpleArray.
XlwTemplate.DataSource("SimpleArray") = SimpleArray

'--- Bind TwoDimArray to the set of
'--- %%=$TwoDimArray data markers.
XlwTemplate.DataSource("TwoDimArray") = TwoDimArray

'--- Bind TwoDimArray2 to the set of
'--- %%=$TwoDimArray2 data markers.
XlwTemplate.DataSource("TwoDimArray2") = TwoDimArray2

'--- Process the template and stream it to the client.
'--- "EmployeeOrders.xls": File name that will appear in the SaveAs dialog
'--- saProcessOpenInExcel: Option to open the workbook in Microsoft Excel
'--- True: Exclude any macros in the template from the generated workbook
XlwTemplate.Process "EmployeeOrders.xls", saProcessOpenInExcel, True
CheckError Err, "An error occurred processing the template."

'--- Close the ExcelTemplate instance
XlwTemplate.Close
```

The first data marker in the template is **%%=$SimpleArray**. **%%=$** indicates that the data source is a simple variable: a 1-dimensional array or a 2-dimensional array. In the ASP code, ExcelTemplate's DataSource property sets the data source for **%%=$SimpleArray** to a 1-dimensional array:

```
Dim SimpleArray(6)
SimpleArray(0) = "Sunday"
SimpleArray(1) = "Monday"
SimpleArray(2) = "Tuesday"
SimpleArray(3) = "Wednesday"
SimpleArray(4) = "Thursday"
SimpleArray(5) = "Friday"
SimpleArray(6) = "Saturday"
XlwTemplate.DataSource("SimpleArray") = SimpleArray
```

The DataSource property's parameter - "SimpleArray" - specifies the name of the template data marker that the data source should populate.

There are three "TwoDimArray" data markers in the template; their fields are specified by ordinal (#1, #2, and #3). The SetDataSource property sets the data source for the three data markers to a 2-dimensional array:

```
Dim TwoDimArray(1, 2) '--- 2-D array
TwoDimArray(0, 0) = "Nancy Davolio"
TwoDimArray(0, 1) = "Sales Manager"
TwoDimArray(0, 2) = "Northwind Sales Dept."
TwoDimArray(1, 0) = "Andrew Suyama"
TwoDimArray(1, 1) = "Analyst"
TwoDimArray(1, 2) = "Northwind Finance Dept."
XlwTemplate.DataSource("TwoDimArray") = TwoDimArray
```

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**.

The last set of data markers in the template includes **%%=$TwoDimArray2.#1**, **%%=$TwoDimArray2.#2**, and **%%=$TwoDimArray2.#3**. The SetDataSource property sets the data source for the three data markers to a 2-dimensional array::

```
Dim TwoDimArray2(0, 2) '--- 2-D array, 1 row
TwoDimArray2(0, 0) = "2001"
TwoDimArray2(0, 1) = "2002"
TwoDimArray2(0, 2) = "2003"
XlwTemplate.DataSource("TwoDimArray2") = TwoDimArray2
```

Data Marker	Code

%%=$SimpleArray	`Dim SimpleArray(6)`
	`SimpleArray(0) = "Sunday"`
	`SimpleArray(1) = "Monday"`
	`SimpleArray(2) = "Tuesday"`
	`SimpleArray(3) = "Wednesday"`
	`SimpleArray(4) = "Thursday"`
	`SimpleArray(5) = "Friday"`
	`SimpleArray(6) = "Saturday"`
	`XlwTemplate.DataSource("SimpleArray") = SimpleArray`

%%=$TwoDimArray.#1
%%=$TwoDimArray.#2
%%=$TwoDimArray.#3
	`Dim TwoDimArray(1, 2) '--- 2-D array`
	`TwoDimArray(0, 0) = "Nancy Davolio"`
	`TwoDimArray(0, 1) = "Sales Manager"`
	`TwoDimArray(0, 2) = "Northwind Sales Dept."`
	`TwoDimArray(1, 0) = "Andrew Suyama"`
	`TwoDimArray(1, 1) = "Analyst"`
	`TwoDimArray(1, 2) = "Northwind Finance Dept."`
	`XlwTemplate.DataSource("TwoDimArray") = TwoDimArray`

%%=$TwoDimArray2.#1 **%%=$TwoDimArray2.#2** **%%=$TwoDimArray2.#3**	

```
Dim TwoDimArray2(0, 2) '--- 2-D array, 1 row
TwoDimArray2(0, 0) = "2001"
TwoDimArray2(0, 1) = "2002"
TwoDimArray2(0, 2) = "2003"
XlwTemplate.DataSource("TwoDimArray2") =
TwoDimArray2
```

Using a Database as a Data Source

The following example demonstrates binding data sources to data markers in the template EmployeeOrdersTemplate.xls. The generated spreadsheet is a sales report for a selected employee. The template contains the following data markers: **%%=Employee.Name**, **%%=Employee.Title**, **%%=Orders.OrderID**, **%%=Orders.Customer**, **%%=Orders.OrderDate**, and **%%=Orders.OrderTotal**. The two sets of data markers - **%%=Employee.*** and **%%=Orders.*** will be populated by two ADO RecordSets.

To get an ADO RecordSet to assign as a data source:

1. Create an ADODB.Connection object to open a database connection.
2. Create a SQL query to get data from the database.
3. Execute the SQL query to return a RecordSet.

   For example, the following lines connect to the database and execute SQL statements to return RecSet and EmpRecSet, the RecordSets that will bind to the **%%=Orders.*** and **%%=Employee.*** data markers in the template:

```
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("connstring")
CheckError Err, "Could not open database connection."

'--- Fill a recordset with orders for the selected employee
SQL = "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="
 & EmployeeId
Set RecSet = Conn.Execute(SQL)
CheckError Err, "Could not query database for sales info."

'--- Fill a recordset with information about the selected employee
Set EmpRecSet = Conn.Execute("SELECT FirstName + ' ' + LastName As Name," & _
 " Title  FROM Employees WHERE EmployeeID=" & EmployeeID)
CheckError Err, "Could not query database for employee info."
```

   To bind a RecordSet to a template data marker, set ExcelTemplate's DataSource property to the RecordSet. Pass the data marker name to DataSource as a parameter to let ExcelWriter know which marker the RecordSet should bind to. For example, the following sets the data source for the data markers **%%=Orders.OrderID**, **%%=Orders.Customer**, **%%=Orders.OrderDate**, and **%%=Orders.OrderTotal** to the Recordset RecSet, and the data source for the data markers **%%=Employee.Name** and **%%=Employee.Title** to EmpRecSet:

```
Set XlwTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")
XlwTemplate.Open Application("templatepath") & _
 "\DataBinding\EmployeeOrdersTemplate.xls"
CheckError Err, "There was an error opening the template workbook."

'--- Bind the sales data recordset to the template datamarkers %%=Orders.*
XlwTemplate.DataSource("Orders") = RecSet

'--- Bind the employee data recordset to the template datamarkers %%=Employee.*
XlwTemplate.DataSource("Employee") = EmpRecSet
```

Output Options

Using the ExcelTemplate object's Process method, you can:

- Save a spreadsheet to **disk**.
- Stream the spreadsheet to the client and open it **in Excel**.
- Stream the spreadsheet to the client and open it **in the browser**. In this case, Internet Explorer's menus and toolbars will be replaced with Excel's.
- Return the spreadsheet as an ExcelApplication object. The spreadsheet can then be modified using ExcelApplication's objects, methods, and properties. See Passing ExcelTemplate to ExcelApplication.

> This feature is **not** available in ExcelWriterSE.

ExcelTemplate's **Process** method takes three parameters:

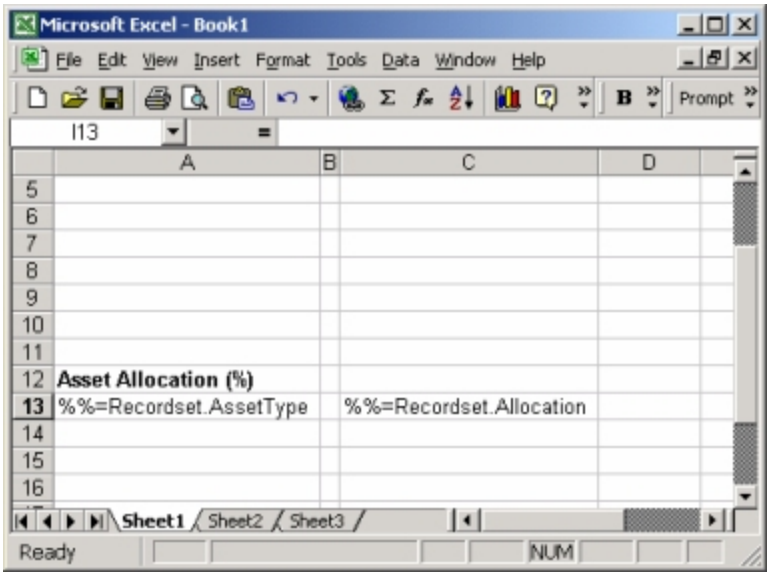
ExcelTemplate.Process Parameters	Description	Default Value
FileName	Path and file name of the new spreadsheet	
ProcessMethod	Optional. Specifies whether to save the file to disk, open it in Excel, open it in the browser, or return an ExcelApplication object. Assign ProcessMethod by name or number: • **0 saProcessDefault** Save the file to disk • **1 saProcessOpenInExcel** Open the file in Excel • **2 saProcessOpenInPlace** Open in the browser • **3 saProcessOpenForScripting** Return the file in memory as an ExcelApplication object. The spreadsheet can then be modified using ExcelApplication's objects, methods, and properties. See How to Use Templates. This value is not available in ExcelWriterSE.	Default value: 0
ExcludeMacro	Optional. If set to True, macros in the template will be excluded from the generated spreadsheet. When ExcludeMacro is enabled, buttons associated with macros may produce the error "Data may have been lost." Therefore, when excluding macros, remove all associated buttons from the spreadsheet.	Default value: False

Templates and Charts

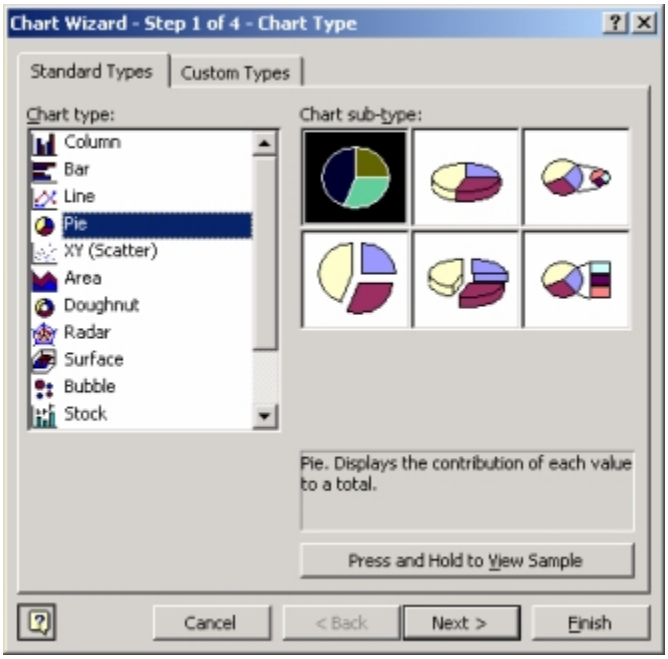
When you include a chart in a template spreadsheet, you can use a data marker column as a data source for the chart. Excel will automatically adjust charts to include the number of rows that ExcelWriter assigns to the data marker column.

How to Link a Chart to a Template Data Marker

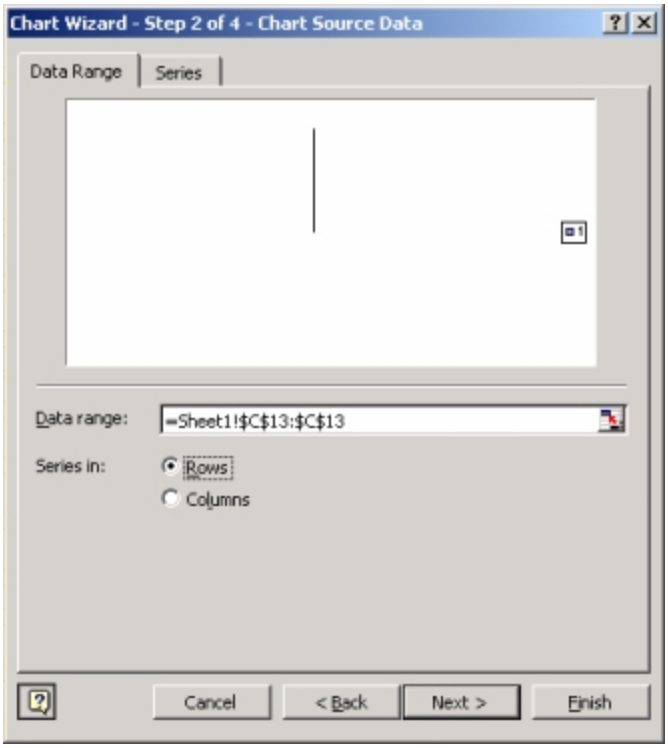
1. In Microsoft Excel, create an ExcelWriter template. For instructions on creating a template, see How to Use Templates.



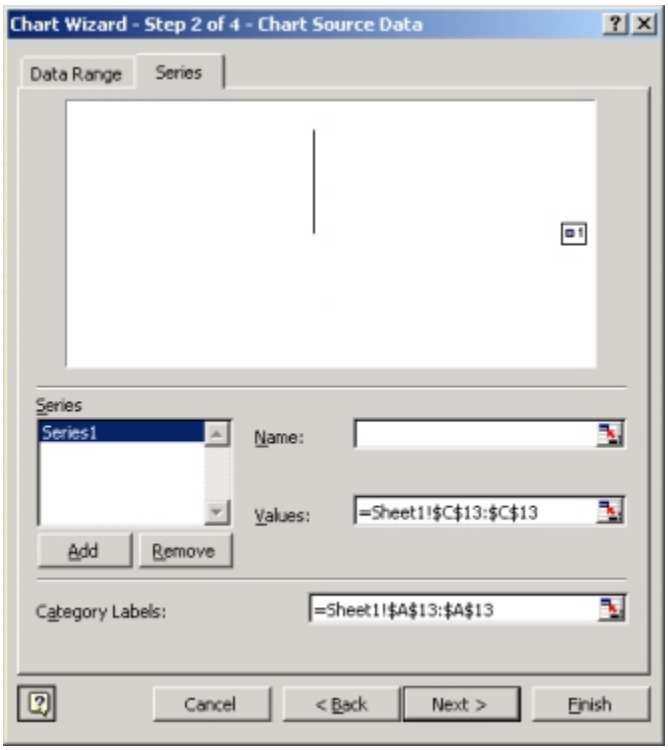
2. Use Excel's Chart Wizard to create a chart in your spreadsheet. Open the **Insert** menu and select **Chart...**, or click the toolbar's Chart Wizard icon. Select the type of chart you would like to create and click **Next**.



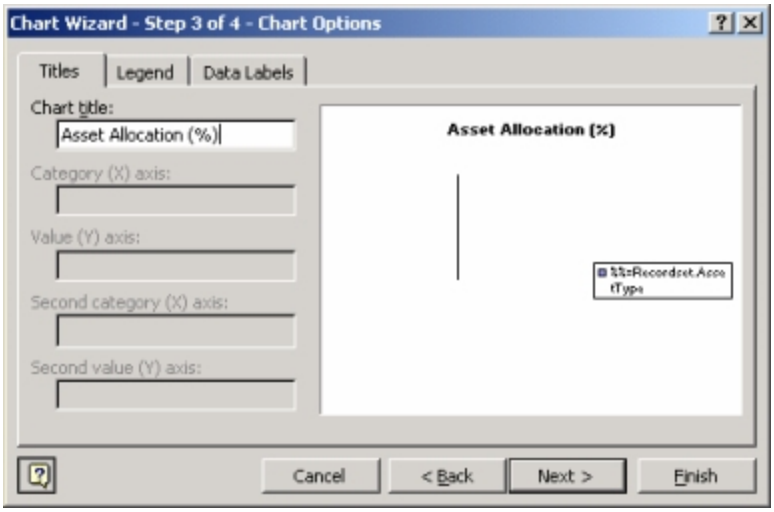
3. In the **Data Range** field, enter the cell that contains the data marker that you want to use as a data source for your chart. Excel requires both a starting point and an ending point in the Data Range Field. So, if your data marker is at cell C13, enter **=Sheet1!$C$13:$C$13**. The number of rows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



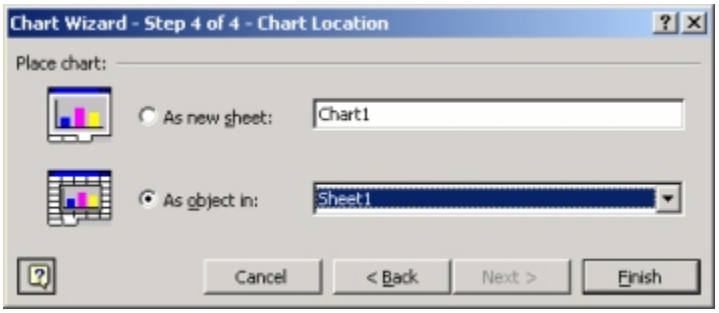
4. Select the **Series** tab, and in the **Category Label** field, enter the range of cells containing the label data marker. Include starting and end points in the category label range. For example, if the label data marker is at cell A13, enter **=Sheet1!$A$13:$A$13** and Excel will automatically adjust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



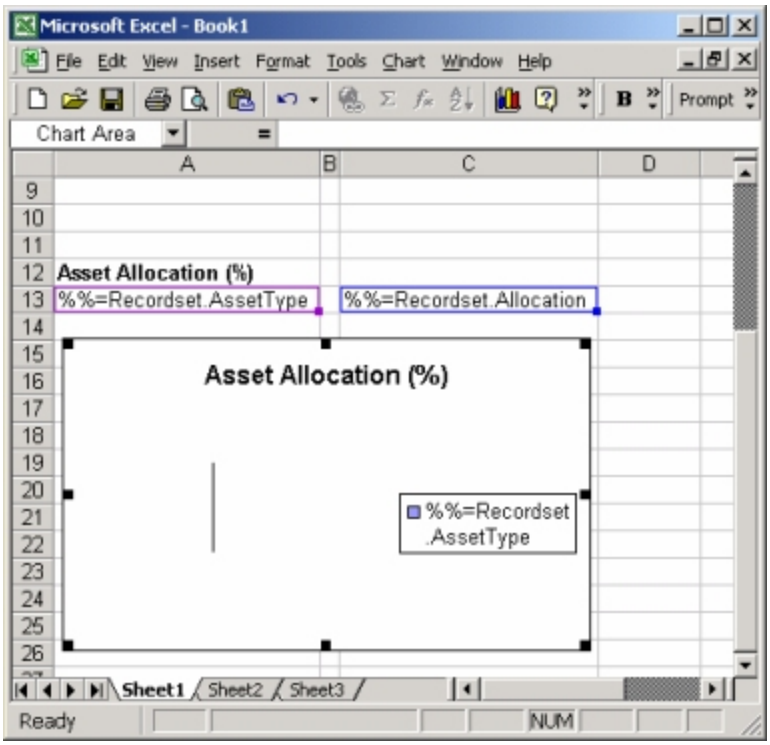
5. Enter a chart title and click **Next**.



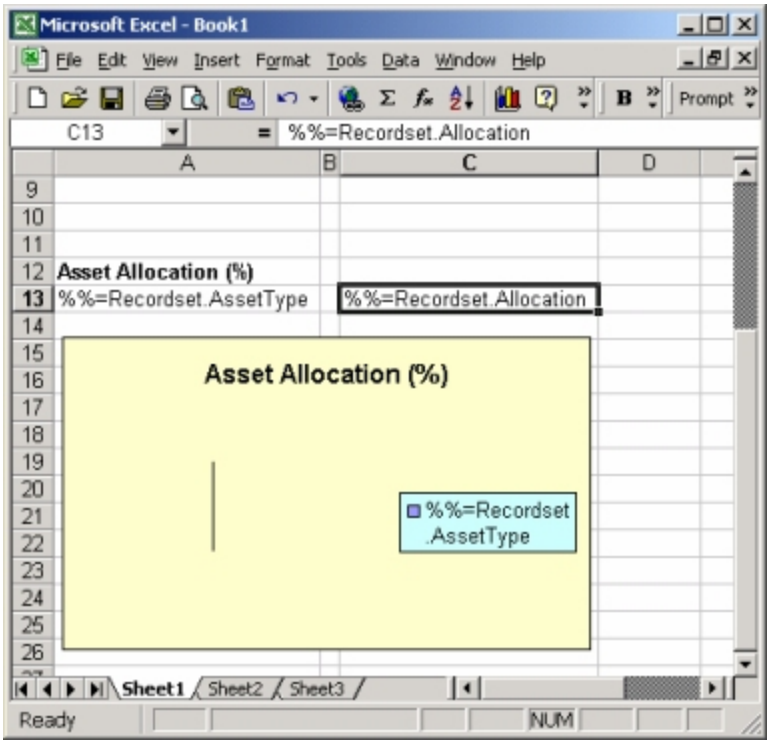
6. Select a chart location and click **Finish** to insert the chart into your spreadsheet.



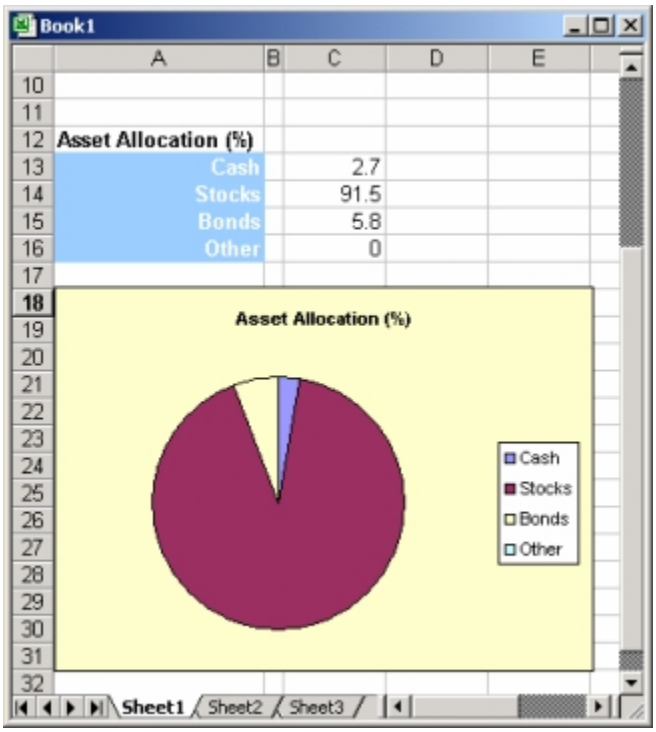
The unpopulated chart is now in your spreadsheet.



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.



Templates and PivotTables

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 in the ExcelWriter-generated spreadsheet.

There are specific guidelines you must follow when creating an ExcelWriter template file with PivotTables.

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 column containing the marker. At runtime, ExcelWriter gets values 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

If you place data markers in the worksheet which will contain the imported data and then create a PivotTable referencing these cells, the data markers will be pulled into the PivotTable. If the PivotTable remains in this state when ExcelWriter's ExcelTemplate object opens the template file, ExcelTemplate will detect these data markers and attempt to load them into the PivotTable itself.

To ensure that ExcelTemplate supplies the PivotTable with real values rather than the literal data marker strings, you must:

- 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. In Microsoft Excel, create a template file containing data markers (see How to Use Templates).
2. Open the **Data** menu and select **PivotTable and PivotChart Report**.
3. 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.
4. Specify the initial layout and field formatting for the PivotTable. Place the PivotTable in any worksheet in the workbook. (See the Excel documentation for more details about creating your PivotTable.)
5. Before clicking **Finish**, click the **Options** button and make sure that **Refresh on open** is not checked.
6. In your new PivotTable, you will now see the data markers. Replace the data marker in each cell of the table with **temporary data** of the appropriate type and format. For example, if the data will be a phone number, write something like "111-111-1111". From this point on, **do not** refresh your PivotTables.
7. Repeat the steps above for each PivotTable you wish to create.
8. Finish creating your template file and save it as a version which you may edit in the future, for example "template_edit.xls".
9. Save another copy of the template file for production, for example "template_production.xls"
10. In the production copy, right-click a PivotTable and select **Table Options**. Check **Refresh on open**. Repeat this for every PivotTable in your workbook. You must enable refreshing in the production for the newly-loaded data to automatically fill your PivotTables when the ExcelWriter-generated workbook is opened on the client.
11. 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.
12. 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, save a second copy of the file as the production version.

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. This section shows you how to populate a template spreadsheet with the ExcelTemplate object, and pass the spreadsheet to ExcelApplicationto add a chart.

In the following example, ExcelTemplate opens the template SalesByCountry.xls, populates it, and passes it to ExcelApplication to be modified. ExcelApplication is used to add a chart to the template. The values imported from the database provide the source data for the chart.

```
SalesDataSql = "SELECT DISTINCT Customers.Country, " & _
 "Sum(([UnitPrice]*[Quantity]*(1-[Discount])/100)*100) AS " & _
 "[Total Sales] FROM (Customers INNER JOIN ([Order Details] " & _
 "INNER JOIN Orders ON [Order Details].OrderID = Orders.OrderID) " & _
 "ON Customers.CustomerID = Orders.CustomerID) " & _
 "GROUP BY Customers.Country ORDER BY Customers.Country"
'--- Create and open an ADO Connection.
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("connstring")

'--- Query the database and get a RecordSet.
Set RecSet = Conn.Execute(SalesDataSql)
Set XlwTempl = Server.CreateObject("SoftArtisans.ExcelTemplate")

'--- Open the template workbook.
XlwTempl.Open("C:\Templates\SalesByCountry_NoChart.xls")

'--- Set PreserveStrings=true to make Excel handle
'--- numeric strings as text values.
XlwTempl.PreserveStrings = True

'--- Set the data source for the template, binding
'--- the DataTable to the Sales data markers
'--- %%=Sales.*
XlwTempl.DataSource("Sales") = RecSet

'--- The Process method enters the RecordSet values in the template
'--- and generates a new spreadsheet. To pass the generated spreadsheet
'--- to ExcelApplication, the second parameter of
'--- Process - ProcessMethod - is set to saProcessOpenForScripting.
Set XlwApp = XlwTempl.Process("", saProcessOpenForScripting)

'--- Add a chart the workbook.
Set ws2 = XlwApp.Worksheets(2)
ws2.Name = "Chart"
Set Chart1 = ws2.Charts.Add(saxlsColumnChart, 0, 1, 1, 21, 12)
Chart1.AutoScale = False

'--- Use he values imported to the first worksheet - named "Data" - to
'--- plot a data series in the chart, and specify the range of category axis
'--- values.
Set Series1 = Chart1.SeriesCollection.Add("=Data!B2:B22", , 1)
Chart1.SeriesCollection.CategoryData = "=Data!A2:A22"

'--- Add a legend to the chart, and assign legend properties.
Chart1.ShowLegend = True
Chart1.LegendLocation = saxlsObjectRight
Chart1.Legend.TextFont = LegendFont
Chart1.Legend.Area.ForegroundColor = RGB(255, 255, 204)

'--- Call ExcelApplication.Save to generate the new spreadsheet
'--- and stream it to the client.
Response.Clear
XlwApp.Save "SalesReportWithChart.xls", saOpenInExcel
Response.End
```

ExcelApplication in Depth

- Addressing Cells
- Setting Values
- Adding a Formula

- Adding Formatting to a Spreadsheet
- ExcelApplication Output Options
- Preserving Leading Zeros in Numeric Strings
- Reading an Existing Spreadsheet
- Modifying an Existing Spreadsheet
- Importing Data from a Database
- Adding Charts
- Creating a Complex Chart
- Creating a PivotTable
- The Range Object
- Page Setup
- Formatting Headers and Footers
- Protecting Your Worksheet
- Multilingual Support
- Passing ExcelApplication to ExcelTemplate

Addressing Cells

How to Address Cells

Cells can be addressed in two ways: by **Name** or by **Number**. Accessing by name is easier to read, more intuitive and matches the cell references seen in Microsoft Excel. Accessing by number is significantly faster, and is easier to program if it is necessary to walk a number of cells.

```
Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set cells = xlw.Worksheets(1).Cells

'--- By name
cells("B3").Value = 123

'--- Same cell by number
cells(3, 2).Value = 123
```

	Addressing Cells by Name	**Addressing Cells by Number**
Format	Cells("*ColLetterRowNumber*")	Cells(*row, column*)
Example	Cells("D8")	Cells(8, 4)
Advantages	Easy to read, intuitive, matches MS Excel cell references	Faster, easier to use when iterating over a set of cells

Example 1: Addressing Cells by Name and Number

When referencing a cell by number, the format is cells(row, column). It is easy to mix and match different means of referencing a cell within your application.

```
Dim oXLW
Dim oCells

'--- Create an instance of ExcelApplication and get a reference
'--- to the first worksheet's cells.
Set oXLW = Server.CreateObject("SoftArtisans.ExcelWriter")
Set oCells = oXLW.Worksheets(1).Cells

'--- Reference a cell by its Excel address.
oCells("B3").Value = 123

'--- Reference a cell using row/column addressing.
oCells(7, 1).Value = 123

oXLW.Save "address1.xls", saOpenInPlace

Set oXLW = Nothing
```

Example 2: Iterating over Cells

It is much easier and faster to iterate over a set of cells by number.

```
Dim xlw, cells
Dim row, column, letterA, cellname

Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set cells = xlw.Worksheets(1).Cells

'--- Address cells by number
For row = 1 To 5
  For column = 1 To 5
    cells(row, column).Value = CStr(row) & ", " & CStr(column)
  Next ' column
Next ' row

For row = 1 To 5
  For column = 27 To 31
    cells(row, column).Value = CStr(row) & ", " & CStr(column)
  Next ' column
Next ' row

'--- Address cells by name.
'--- This works only up to column Z. It will not work for column AA.
letterA = Asc("A") ' ASCII numerical value of the letter A
For row = 7 To 12
  For column = 1 To 5
    cellname = Chr(column + letterA - 1) & CStr(row)
    cells(CStr(cellname)).Value = cellname
  Next ' column
Next ' row

'--- This is more complicated but works for any column
'---
For row = 7 To 12
  For column = 27 To 31
    cellname = ""
    If (column mod 26) <> 0 Then
      If (column \ 26) > 0 Then cellname = Chr((column \ 26) + letterA - 1)
      cellname = cellname & Chr((column Mod 26) + letterA - 1) & CStr(row)
    Else
      cellname = chr(column + letterA - 1) & cStr(row)
    End If
    cells(CStr(cellname)).Value = cellname
  Next ' column
Next ' row

For row = 14 To 18
  For column = 1 To 5
    cellname = ""
    If (column mod 26) <> 0 Then
      If (column \ 26) > 0 Then cellname = Chr((column \ 26) + letterA - 1)
      cellname = cellname & Chr((column Mod 26) + letterA - 1) & CStr(row)
    Else
      cellname = chr(column + letterA - 1) & cStr(row)
    End If
    cells(CStr(cellname)).Value = cellname
  Next ' column
Next ' row

'--- For a specific cell, addressing by name is easier
cells("G1").Value = "Please scroll to cell AA1->>>"
cells("G1").Format.Font.Bold = True

xlw.Save "address2.xls", saOpenInPlace

Set xlw = Nothing
Response.End
```

Setting Values

Methods for Assigning Cell Values

Cell values can be assigned:

- Programmatically, on a cell-by-cell basis with data that has been parsed from a CSV file or an XML file
- By iterating through an ADO Recordset and assigning field values to cells on a row-by-row basis
- By importing a recordset in a single step with CopyFromRecordset or GetDataFromRecordset
- By importing a recordset in a single step using ExcelTemplate
- By copying data from an existing Excel file into the newly created Excel sheet using GetDataFromExcelSheet
- By opening an existing Excel file, and saving it with a new name

ExcelWriter allows you to format these imported cell values as virtually any data type that is supported by Excel. This includes currency, date/time, numeric data types such as integer or floating point number, strings, text, etc. Casting - changing cell values from one data type to another - can be accomplished by means of the server-side scripting functions (i.e. VBScript's CStr() or CINT() functions).

Example 1: Data Types

Microsoft Excel supports many different types of data in a cell: integer and floating point numbers, text, date/time, currency. The following example shows a broad range of different data types.

```
Dim xlw
Dim cells

Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set cells = xlw.Worksheets(1).Cells

'--- Assign some values and styles
cells("A1").Value = "Type"
cells("B1").Value = "Value"

'--- Integer value
cells("A2").Value = "Integer"
cells("B2").Value = 123

'--- Float value
cells("A3").Value = "Float"
cells("B3").Value = 1.05e12

'--- String value
cells("A4").Value = "String"
cells("B4").Value = "some text"

'--- Date value using VBScript's CDate() function
cells("A5").Value = "Date(ASP)"
cells("B5").Value = CDate("1/1/2000")

'--- Large integer
cells("A8").Value = "Big integer"
cells("B8").Value = 999999999999999

'--- Currency
cells("A9").Value = "Currency"
cells("B9").Value = "$5000"

'--- Negative integer
cells("A10").Value = "Neg. Integer"
cells("B10").Value = "-2"

'--- Negative floating point
cells("A11").Value = "Neg. Float"
cells("B11").Value = "-2.5e-23"

'--- Open the spreadsheet in the browser
xlw.Save "value1.xls", saOpenInPlace

'--- Clean up
Set xlw = Nothing
Response.End
```

Example2: Changing from One Type to Another

The previous example showed how to enter values with default interpretation by ExcelWriter. It is also possible to explicitly convert one type into another - a process known as **casting**.

```
Dim xlw, cells

Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set cells = xlw.Worksheets(1).Cells

cells("A1").Value = "Cast"
cells("B1").Value = "Value"

'--- You can use VBScript's casting functions
'--- to assign strings as integers and vice versa
cells("A2").Value = "String to Integer"
cells("B2").Value = CInt("123")

cells("A3").Value = "Integer to String"
cells("B3").Value = CStr(123)

'--- Use VBScript's CCur() to cast numbers to currency
cells("A4").Value = "Currency"
cells("B4").Value = CCur(1000000.45)

'--- Open spreadsheet in the browser
xlw.Save "value2.xls", saOpenInPlace

'--- Clean Up
Set xlw = Nothing
Response.End
```

Adding a Formula

How to Add a Formula

ExcelWriter supports almost all major math, string, boolean, time, statistical, database, lookup and reference formulas or functions that are part of Excel. Formulas can be inserted into any cell in a worksheet by using the Cell.Formula property. They can be inserted either by hard coding the cell address or by dynamic coding using a variable, such as NumRows (the return value of the CopyFromRecordset method which indicates where the last row of the pasted recordset will be positioned in the worksheet) to provide a sum at the bottom of a column.

In ExcelWriter, formulas can reference data across worksheets, as follows:

```
ws3.Cells(1, 1).Formula = "=sum(sheet1!A1,Sheet2!A2)"
ws3.Cells(2, 1).Formula = "=sum(sheet1!A1:A3)"
```

When applying a formula to a cell, always include the entire string in double quotation marks ("") and begin the string with an equal sign (=), as you would when creating a spreadsheet in Microsoft Excel.

ExcelWriter does not execute formulas on the server. Therefore, you cannot retrieve a formula's result in script. All formulas are executed when the file is opened in Microsoft Excel.

If a downloaded spreadsheet is opened with Microsoft Excel Viewer, "0" will be displayed in all cells that contained formulas.

The only functions available in ExcelWriterSE are sum and average. ExcelWriterEE includes all functions listed in Formula Functions.

Example 1: Adding Formulas

Formulas are added using the Cell object's Formula property. The Formula property is used to explicitly say that this is a formula to be interpreted rather than literal text. The formula is specified as a simple text string. Most of the commonly used functions in Microsoft Excel are supported by ExcelWriter. For the complete list of the currently supported functions, see see the Formula Functions.

To specify a range of cells, use the format starting-cell:ending-cell. Many more complex combinations of functions can be specified in a formula.

```
Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set ws = xlw.Worksheets(1)

'--- Create header row
ws.Cells("A1").Value = "Name"
ws.Cells("B1").Value = "Count"
ws.Cells("C1").Value = "Dollar"

'--- Add values
ws.Cells("A2").Value = "Fred Smith"
ws.Cells("B2").Value = 10
ws.Cells("C2").Value = 37.5

ws.Cells("A3").Value = "John Doe"
ws.Cells("B3").Value = 20
ws.Cells("C3").Value = 62.85

'--- Add formulas
ws.Cells("B5").Value = "Total:"
ws.Cells("B6").Formula = "=sum(b2:b3)"

ws.Cells("C5").Value = "Average:"
ws.Cells("C6").Formula = "=average(c2:c3)"

ws.Cells("E5").Value = "Total x Average:"
ws.Cells("E6").Formula = "=b6*c6"

'--- Open the workbook in the browser
xlw.Save "formula.xls", saOpenInPlace

Set xlw = Nothing
Response.End
```

Example 2: Accessing Values in a Different Worksheet

ExcelWriter allows you to use values from a different sheet when assigning a formula to a cell.

ExcelWriter does **not** support the following formulations:

- Sheet1:Sheet2!A1:C3
- Sheet1!A1:C3;Sheet4!A1:C3

```
Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set ws = xlw.Worksheets(1)

'--- Create header row
ws.Cells("A1").Value = "Name"
ws.Cells("B1").Value = "Count"
ws.Cells("C1").Value = "Dollar"

'--- Add values
ws.Cells("A2").Value = "Fred Smith"
ws.Cells("B2").Value = 10
ws.Cells("C2").Value = 37.5

ws.Cells("A3").Value = "John Doe"
ws.Cells("B3").Value = 20
ws.Cells("C3").Value = 62.85

'--- Add formulas
ws.Cells("B5").Value = "Total:"
ws.Cells("B6").Formula = "=sum(b2:b3)"

ws.Cells("C5").Value = "Average:"
ws.Cells("C6").Formula = "=average(c2:c3)"

ws.Cells("E5").Value = "Total x Average:"
ws.Cells("E6").Formula = "=b6*c6"

Set ws = Nothing

Set ws2 = xlw.Worksheets(2)
ws2.Cells("B5").Value = "Total:"
ws2.Cells("B6").Formula = "=sum(Sheet1!b2:b3)"

ws2.Cells("C5").Value = "Average:"
ws2.Cells("C6").Formula = "=average(Sheet1!c2:c3)"

ws2.Cells("E5").Value = "Total x Average:"
ws2.Cells("E6").Formula = "=Sheet1!b6*Sheet1!c6"

'--- Open the workbook in the browser
xlw.Save "formula.xls", saOpenInPlace

Set xlw = Nothing
Response.End
```

Adding Formatting to a Spreadsheet

Style, Format, and Font

Individual cells can be formatted by using a combination of **Styles**, **Formats** and **Fonts**.

A **style** ensures that a set of cells (for instance those that are contained in a predefined Range object) have the same look. If the style is changed, all cells that reference that particular style will automatically have the updated look.

Like a style, a **format** is used to set the look of text, defining characteristics such as alignment, text wrap, and so on. However, a format is specific to **one cell**. Updating a cell's format will not automatically update the look of other cells. Updating a style will automatically update all cells where the style was applied.

Once a style is applied to a cell, it is possible to supplement that style with the addition or modification of certain properties by using Cell.Format. However, this should be done cautiously. See Applying Formatting Efficiently.

A **font** is always the property of a style or format. Use Font to define text properties like size, color, italic, and font name.

Example: Adding Formatting to a Spreadsheet

In the following example, styles are applied to individual styles. You can also apply a style to range of cells. For more information, see Range.Style.

```
'--- Declarations
Dim oXLW, oWS
Dim oMyFont, oNumStyle, oTextStyle, oHeadingStyle,
Dim oBorderStyle, oBGColorStyle, oWrapStyle

Set oXLW = Server.CreateObject("Softartisans.ExcelWriter")
Set oWS = oXLW.Worksheets(1)

'--- Create header row
oWS.Cells("A1").Value = "Name"
oWS.Cells("B1").Value = "Count"
oWS.Cells("C1").Value = "Dollar"

'--- Add values
oWS.Cells("A2").Value = "Fred Smith"
oWS.Cells("B2").Value = 10
oWS.Cells("C2").Value = 37.5

oWS.Cells("A3").Value = "John Doe"
oWS.Cells("B3").Value = 20
oWS.Cells("C3").Value = 62.85

'--- Add formulas
oWS.Cells("B5").Value = "Total:"
oWS.Cells("B6").Formula = "=sum(b2:b3)"

oWS.Cells("C5").Value = "Average:"
oWS.Cells("C6").Formula = "=average(c2:c3)"

oWS.Cells("E5").Value = "Total x Average:"
oWS.Cells("E6").Formula = "=b6*c6"

'--- Create a font
Set oMyFont = oXLW.CreateFont
oMyFont.Size = 10
oMyFont.Italic = True
oMyFont.Name = "Tahoma"

'--- Create styles
Set oNumStyle = oXLW.CreateStyle
oNumStyle.Font = oMyFont
oNumStyle.HorizontalAlignment = sahaRight

Set oTextStyle = oXLW.CreateStyle
oTextStyle.Font = oMyFont
oTextStyle.HorizontalAlignment = sahaLeft

Set oHeadingStyle = oXLW.CreateStyle
oHeadingStyle.Font = oMyFont
oHeadingStyle.Font.Size = 12
oHeadingStyle.Font.Bold = True
oHeadingStyle.HorizontalAlignment = sahaCenter

'--- A boxed border around a cell
Set oBorderStyle = oXLW.CreateStyle
oBorderStyle.BorderLineStyle = 6

'--- A Blue background color
Set oBGColorStyle = oXLW.CreateStyle
oBGColorStyle.BackgroundColor = RGB(0, 0, 255)

'--- Word-wrap style
Set oWrapStyle = oXLW.CreateStyle
oWrapStyle.WrapText = True
```

```
'--- Apply styles to cells
oWS.Cells("A1").Style = oHeadingStyle
oWS.Cells("B1").Style = oHeadingStyle
oWS.Cells("C1").Style = oHeadingStyle

oWS.Cells("A2").Style = oTextStyle
oWS.Cells("A3").Style = oTextStyle

oWS.Cells("B2").Style = oNumStyle
oWS.Cells("B3").Style = oNumStyle

'--- Define a format for a single cell
oWS.Cells("E6").Format.Number = 2
oWS.Cells("E6").Format.Font.Name = "Courier New"
oWS.Cells("E6").Format.Font.Color = 255 '---RGB Value

'--- Demonstrate the Border capability
oWS.Cells("B10").Value = "Boxed in!"
oWS.Cells("B10").Style = oBorderStyle

'--- Demonstrate Background colorization
oWS.Cells("D10").Value = "BGColor"
oWS.Cells("D10").Style = oBGColorStyle

'--- Wordwrap text inside a cell
oWS.Cells("B12").Value = "Hello this is a string that will be wrapped."
oWS.Cells("B12").Style = oWrapStyle
oWS.Cells.Range(12, 2, 1, 1).AutoFitHeight

'--- Open the workbook in the browser
oXLW.Save "Format.xls", saOpenInPlace
```

```
Set oXLW = Nothing
Response.End
```

Applying Formatting Efficiently

ExcelWriter allows you to apply an unlimited number of formats and styles to a workbook. However, Microsoft Excel supports up to 4074 unique formats and styles in a single workbook. If this limit is exceeded in an ExcelWriter-generated spreadsheet, ExcelWriter will throw an error when the Save method is called. Therefore, when designing the layout of the workbook, it is essential to carefully plan the application of formats and styles to prevent unnecessary overlapping of these.

Styles are "shared objects" while formats are unique for each assignment. This means that application of a style to more than one cell counts as a single instance of unique formatting, but the application of a format to a cell counts as a single instance for **each** application. Therefore, styles should be used whenever possible and formats for specific cells only when required to supplement a style.

Avoid the application of styles to non-essential cells within an ExcelWriter-generated workbook. To accomplish this, narrow the cell range by setting all four optional parameters (FirstRow, FirstCol, NumRows, NumCols) as opposed to relying on the defaults. This prevents the creation of excessive cell formatting records that can increase file size and thereby affect the performance of creating and saving the workbook.

Keep in mind that since a style is shared, once it is assigned to a range, any subsequent modifications of the style will update the look of all cells in the range. This is often not the intention of developers and it can be a source of confusion when the output is not as intended.

Note that starting with Version 4, ExcelWriter saves only unique fonts, formats, and styles. This reduces the size of generated spreadsheets, improves performance, and prevents exceeding Excel's limit.

ExcelApplication Output Options

After creating a workbook with ExcelApplication, you can:

- Save it to disk
- Return it as a sequence of bytes in memory.
- Stream it to the client and open it in Excel.
- Stream it to the client and open it in the browser. In this case, Internet Explorer's menus and toolbars will be replaced with Excel's.
- Return it as an **ExcelTemplate** object.

ExcelApplication's **Save** method takes three optional parameters:

```
[varBytes =] objExcelApplication.Save [FileName], [SaveMethod], [SaveFormat]
```

ExcelApplication.Save Parameters

FileName	Optional. If saving to disk, assign the path and file name of the new spreadsheet. If opening the file, assign a file name only. To return the file in memory, do not assign a value to this parameter.
SaveMethodDefault value: 0	Optional. Specifies whether to save the file to disk, open it in Excel, open it in the browser, return it in memory, or return it as an ExcelTemplate object. • **0 saDefault** If a FileName is assigned, the file will be saved to disk. If a FileName is **not** assigned, the file will be returned in memory as a sequence of bytes. • **1 saOpenInExcel** Open the file in Excel • **2 saOpenInPlace** Open in the browser • **3 saOpenAsTemplate** Return an ExcelTemplate object.
SaveFormatDefault value: 8	Optional. Specifies whether to save the spreadsheet in BIFF7 (Excel 95) or BIFF8 (Excel 97/2000) format. • **8 saFileFormatDefault** • **7 saFileFormatExcel95**

To use ExcelWriter constants (e.g., saFileFormatExcel95), include ExcelWriter's TypeLib metadata tag at the beginning of your script:

```
<!--METADATA TYPE="TypeLib" UUID="{7BCD2133-64A0-4770-843C-090637114583}"-->
```

ExcelApplication.Save Examples

Example	Result
`varBytes = xlw.Save`	Return the spreadsheet as a sequence of bytes to varBytes.
`xlw.Save "d:\reports\report1.xls", , saFileFormatExcel95`	Save the file in Excel95 (BIFF7) format to a specific location on the hard disk.
`xlw.Save "report.xls", saOpenInExcel`	Open in Excel on the client. If the user saves the file, it will have report.xls as the default name. If the file has already been cached by the browser, it will appear as report(1).xls.
`xlw.Save "report.xls", saOpenInPlace`	Open in the browser window. If the user doesn't have MS-Excel support in IE, it will ask the user to save the file. In this case, the Filename property isn't used; instead the browser will use the webserver script(which generates this Excel file) name as the default name.

Preserving Leading Zeros in Numeric Strings

When ExcelWriter is used to add a numeric string value to a single cell or to add numeric strings to an entire column in a worksheet, these values will be displayed in Excel's "General" format unless otherwise specified. This format will align values to the right and trim leading zeros. If, for example, you enter zip codes in a column of cells when the cell format is "General," leading zeros will be lost.

The recommended method for retaining leading zeros in numeric strings is to format cells that contain numeric strings as text. This can be accomplished by setting:

- Cell.Format.Number to 49 or @,
- Style.Number to 49 or @
- Cells.PreserveStrings to true.

If you set Cells.PreserveStrings to true, all numeric strings in the generated spreadsheet will be formatted as text. To format an individual cell value as text, set the cell's Style.Number or Format.Number property to 49 or @.
Here is an example which demonstrates how to use Format.Number to retain the leading zero in a single cell ("A1"):

```
Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set ws = xlw.Worksheets(1)

ws.Cells("A1").Value = "012345"

'--- Set Format.Number to "@" to format the cell
'--- value as text.  Alternatively set Format.Number
'--- to 49.
ws.Cells("A1").Format.Number = "@"
```

Here is an example which demonstrates how to use Style.Number to retain the leading zeros in a whole column (A or 1) of cells:

```
Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set ws = xlw.Worksheets(1)

'--- Create a TextStyle which will retain leading "0"s in
'--- numeric strings.
Set TextStyle = xlw.CreateStyle

'--- Set TextStyle's Number property to "@" to format the cell
'--- values as text.  Alternatively set Format.Number
'--- to 49.
TextStyle.Number = "@"

FirstRow = 1
FirstCol = 1

'--- Paste the Recordset into the Worksheet.
NumRows = cells.CopyFromRecordset(oRs, True, FirstRow, FirstCol)
For r = FirstRow + 1 to NumRows + 1
 Cells(r, FirstCol).Style = TextStyle
Next
```

Reading an Existing Spreadsheet

With Cells.GetDataFromExcelSheet you can read data from an existing Excel workbook and copy it to another workbook. GetDataFromExcelSheet can be called more than once in the process of creating a new workbook.

> GetDataFromExcelSheet can only retrieve cell values. To retrieve charts, formulas, macros, and VBA, use ExcelTemplate or ExcelApplication.Open.

GetDataFromExcelSheet takes seven parameters and returns a Range object:

```
Range = Cells.GetDataFromExcelSheet(FileName, [Source], [IncludeFormatting]
 [FirstRow], [FirstCol], [MaxRows], [MaxCols])
```

You can use the Range returned in formulas, as source data for a chart, to update a database via ADO code that parses out the data, or to apply formatting.

Parameter	Definition	Default Value
FileName	The name of the source file, that is, the workbook from which ExcelWriter is reading.	
[Source]	Optional. Specifies the first cell in the source range of cells. *Source* may include both sheet number and cell name (e.g. "Sheet2!$B$11"), or a cell name alone (e.g. "B11"). If *Source* does not include a sheet reference, ExcelWriter will read from the first sheet in the workbook.	"" *If you do not assign a Source value, ExcelWriter will start reading at cell A1 in the first worksheet.*
[IncludeFormatting]	Optional. When set to True, ExcelWriter will copy the original formatting with the data from the existing workbook.	True
[FirstRow]	Optional. First row in the destination sheet.	1
[FirstColumn]	Optional. First column in the destination sheet.	1
[MaxRows]	Optional. Maximum number of rows to write in the destination sheet.	16384 in Excel 95 (BIFF7) 65536 in Excel 97/2000 (BIFF8)
[MaxColumns]	Optional. Maximum number of columns to write in the destination sheet.	256 (Limit imposed by Microsoft's BIFF7 and BIFF8 formats, not by ExcelWriter)

How to Read from a workbook with GetDataFromExcelSheet

To get data from an existing Excel workbook,

1. Use GetDataFromExcelSheet to specify the source file, and, optionally, the first source cell, the first destination cell, and the maximum numbers of rows and columns to copy.
   Example: To get data from C:\workbooks\workbook1.xls, sheet 2, starting at cell B3, without formatting, and copy to cells B3:F12 in the destination file, use

   ```
   Set ResultRange = ws.Cells.GetDataFromExcelSheet("C:\workbooks\workbook1.xls", "Sheet2!$B$3",
   False, 3, 2, 10, 5)
   ```

2. Save the destination file.
   Example:

   ```
   xlw.Save "c:\workbook2.xls"
   ```

Importing Formulas

Formulas contained in a cell that is imported as part of the range object can be copied. However, the cell values that are the result of the execution of the formula will not be available on the server. This means, for example, that if the requirement is to obtain the "sum" of a certain set of cells server-side for reference in an ADO update to a database, the true value will not be accessible and the cell value for the cell containing the formula will read as "0".

Example: Getting Data from an Existing Excel Workbook

The following script gets data from the workbook source.xls.

```
'--- Declarations
Dim oXLW
Dim oWS
Dim SourceFile
Dim ResultRange
Dim ScriptName

Set oXLW = Server.CreateObject("Softartisans.ExcelWriter")

'--- Create a worksheet in the workbook.
Set oWS = oXLW.Worksheets(1)

'--- Assign an existing spreadsheet to a variable.
SourceFile = "C:\reports\source.xls"

'--- Get data from source.xls, sheet 2, starting at cell B3,
'--- with formatting, and copy to cells B2:C5 in the
'--- destination file.
Set ResultRange = oWS.Cells.GetDataFromExcelSheet(SourceFile,_
 "Sheet2!$B$3", True, 2, 2, 4, 2)

'--- Save the destination file.
oXLW.Save "Read.xls", saOpenInPlace

'--- Clean up.
Set oXLW = Nothing
```

Modifying an Existing Spreadsheet

With ExcelWriter you can open and modify an existing spreadsheet and save it with a new name or stream it to the browser (preserving the original file). This allows you to use a preset format for similar spreadsheets, rather than recreate the format for each. Alternatively, use a template . To determine which method is best for you, see ExcelApp.Open vs. ExcelTemplate.

If either saOpenInPlace or saOpenInExcel is used in the Save method and the new spreadsheet is given the same name as the original file that exists on the server, the original file will not be overwritten.

The original spreadsheet can include macros, VBA, charts, etc. - they will be preserved in the new spreadsheet.

When opening an existing spreadsheet, ExcelWriter will not preserve strings that contain more than one font.

> If you open an Excel 95 (BIFF7) spreadsheet with ExcelWriter, ExcelWriter will process and save the file in Excel 95 format. Therefore, you will not be able to add features to the file that are not supported by Excel 95. For example, since an Excel 95 spreadsheet may include up to 16384 rows, if you open an Excel 95 spreadsheet with ExcelWriter, you cannot use ExcelWriter to include more than 16384 rows in the spreadsheet (although ExcelWriter does support up to 65536 rows for Excel 97/2000/XP).

- How to Modify an Existing Spreadsheet with ExcelApplication.Open()
- Example 1: Creating an Order Spreadsheet from a Preset Order Form
- Example 2: Creating an Invoice from a Preset Invoice Form
- Example 3: Modifying the Source Data for Pivot Tables

How to Modify an Existing Spreadsheet with ExcelApplication.Open

To modify an existing spreadsheet

1. Open the spreadsheet using the ExcelApplication object's Open method:

```
Set objExcelApp = Server.CreateObject("SoftArtisans.ExcelWriter")
objExcelApp.Open "c:\folder\file.xls"
```

> ExcelApplication.Open cannot open a file containing data markers.

2. Create at least one Worksheet, for example:

```
Set order = xlw.Worksheets(1)
```

3. Assign cell values, for example:

```
order.Cells("A7").Value = "Margaret Anderson"
order.Cells("A8").Value = "32 Andover Lane"
order.Cells("A9").Value = "Lawrence, NY"
```

4. Save the modified file with a new name. For information on save options, see ExcelApplication Output Options.

Example 1: Creating an Order Spreadsheet from a Preset Order Form

The following example opens the model order spreadsheet orderform.xls, fills in information, and saves the new order as order.xls.

```
Dim oXLW
Dim oWS

Set oXLW = Server.CreateObject("Softartisans.ExcelWriter")

'--- Open the model order spreadsheet, orderform.xls.
oXLW.Open("C:\forms\orderform.xls")

Set oWS = oXLW.Worksheets(1)

'--- Assign cell values.
'--- Shipping Address:
oWS.Cells("A2").Value = "Margaret Anderson"
oWS.Cells("A3").Value = "32 Andover Lane"
oWS.Cells("A4").Value = "Lawrence, NY"

'--- Billing Address:
oWS.Cells("A7").Value = "Margaret Anderson"
oWS.Cells("A8").Value = "32 Andover Lane"
oWS.Cells("A9").Value = "Lawrence, NY"

'--- Payment Method:
oWS.Cells("A12").Value = "credit card"

'--- Credit Card Number:
oWS.Cells("A15").Value = "1111 2222 3333 4444"

'--- Expiration Date:
oWS.Cells("B15").Value = "03/15/03"

'--- Items:
oWS.Cells("A19").Value = "Hat"
oWS.Cells("A20").Value = "Scarf"
oWS.Cells("A21").Value = "Shirt"
oWS.Cells("A22").Value = "Belt"
oWS.Cells("A23").Value = "Pin"

'--- Quantities:
oWS.Cells("B19").Value = "2"
oWS.Cells("B20").Value = "1"
oWS.Cells("B21").Value = "3"
oWS.Cells("B22").Value = "2"
oWS.Cells("B23").Value = "5"

'--- Prices:
oWS.Cells("C19").Value = "10.99"
oWS.Cells("C20").Value = "18.49"
oWS.Cells("C21").Value = "9.99"
oWS.Cells("C22").Value = "8.79"
oWS.Cells("C23").Value = "2.75"

'--- Shipping and Handling:
oWS.Cells("D25").Value = "1.50"

'--- Open the workbook in the browser.
oXLW.Save "Open.xls" , saOpenInPlace

Set oXLW = Nothing
```

Example 2: Creating an Invoice from a Preset Invoice Form

The following example opens the model invoice spreadsheet openinvoice.xls, fills in information, and opens the new invoice in the browser window.

```
Dim oXLW
Dim strFileName
Dim strScriptName

Set oXLW =  Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Open the existing workbook
oXLW.Open Server.MapPath("./OpenInvoice.xls")

'--- Set the values of the cells within the original workbook
oXLW.Worksheets(1).Cells("M3").Value = "sdgt1"
oXLW.Worksheets(1).Cells("D10").Value = "FirstN LastN"
oXLW.Worksheets(1).Cells("D11").Value = "200 Some street"
oXLW.Worksheets(1).Cells("D12").Value = "Watertown"
oXLW.Worksheets(1).Cells("G12").Value = "Massachusettes"
oXLW.Worksheets(1).Cells("I12").Value = "02472"
oXLW.Worksheets(1).Cells("D13").Value = "000-000-0000"

oXLW.Worksheets(1).Cells("N10").Formula ="=Now()"
oXLW.Worksheets(1).Cells("N11").Value = "1"
oXLW.Worksheets(1).Cells("N12").Value = "FDR"

oXLW.Worksheets(1).Cells("C18").Value = "1000"
oXLW.Worksheets(1).Cells("D18").Value = "Party Hats"
oXLW.Worksheets(1).Cells("L18").Value = "1.75"

oXLW.Worksheets(1).Cells("C20").Value = "500"
oXLW.Worksheets(1).Cells("D20").Value = "Belhaven Stout case"
oXLW.Worksheets(1).Cells("L20").Value = "6.75"

'--- Save and stream the workbook
oXLW.Save "OpenInvoice_Result.xls", saOpenInPlace

'--- Clean up
Set oXLW = Nothing
```

Example 3: Modify the Source Data for Pivot Tables

The following example opens OpenPivotTable.xls, modifies the source data in sheet 2, and saves the new file as OpenPivotTable_Result.xls.

```
Dim oXLW, oWS
Dim strFileName
Dim strScriptName

'--- The path to the file we'll open for editing
strFileName = "C:\Reports\OpenPivotTable.xls")

'--- Create an instance of the ExcelApplication object,
'--- representing your Excel workbook.
Set oXLW = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Open the spreadsheet OpenPivotTable.xls.
oXLW.Open strFileName

'--- Get a reference to the second worksheet in the workbook
Set oWS = oXLW.Worksheets(2)

'--- Assign cell values
oWS.Cells("C2").Value = 307
oWS.Cells("D2").Value = 418
oWS.Cells("E2").Value = 854
oWS.Cells("F2").Value = 500

'--- Save and stream the edited workbook to the client
oXLW.Save "OpenPivotTable_Result.xls", saOpenInPlace

'--- Clean up
Set oXLW = nothing
```

Importing Data from a Database

When you import cells from a database you can apply formatting and add formulas before or after the import. The imported RecordSet can be placed in any location in the spreadsheet. You can import multiple RecordSets to different locations in the spreadsheet.

The first step is quite simple. A standard ADO RecordSet is created based upon a query to the database.

```
Set oConn = Server.CreateObject("ADODB.Connection")
oConn.Open "DBQ=" & filePath & _
  ";Driver= {MicrosoftAccessDriver(*.mdb)};DriverId=25;FIL=MSAccess;"
Set oRs = oConn.Execute ("select * from qryOrders" )
```

An instance of ExcelWriter is created. Database import is performed using the Cells object.

```
Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set cells = xlw.Worksheets(1).Cells
```

After setting up formats, the FirstRow and FirstCol are set. An ADO RecordSet can be considered a rectangular block of data with rows and columns. ExcelWriter can import the RecordSet into any rectangular area on the WorkSheet. In this case, the RecordSet should be placed at Cell (3, 2) which is cell B3.

```
FirstRow = 3
FirstCol = 2
```

The import is performed using the CopyFromRecordSet method. CopyFromRecordSet returns the total number of rows imported into the WorkSheet and fills in the headers automatically.

```
NumRows = cells.CopyFromRecordset(oRs, True, FirstRow, FirstCol)
```

The CopyFromRecordSet method has the following parameters:

```
NumRows = cells.CopyFromRecordset(ADO-RecordSet, [ShowHeaders (True/False)],_
 [FirstRow], [FirstColumn], [MaxRows], [MaxColumns],_
 [FieldList], [FieldListInclude (saxlsExclude/saxlsInclude)])
```

ShowHeaders is true by default, meaning that the first row will contain the column headers retrieved from ADO. FirstRow and FirstColumn have a default value of 1. So, by default the RecordSet will be imported into cell A1. MaxRows has a default value of 65,536 for BIFF8, and 16,384 for BIFF7. MaxColumns has a default value of 256. These limits are imposed by Microsoft's BIFF8 and BIFF7 format and are not a limitation of ExcelWriter. The FieldList parameter is an empty string by default, so FieldListInclude is then saxlsExclude by default.

By default, the CopyFromRecordSet method imports all columns in the recordset. To exclude unnecessary columns from being imported, use the FieldList parameter. For example, the following will include only columns 1, 2, and 3 of the recordset.

```
NumRows = Cells.CopyFromRecordset(oRs, True, 1, 1, 100, 100, "1,2,3", saxlsInclude)
```

Alternatively, the following will exclude columns 4 and 5.

```
NumRows = Cells.CopyFromRecordset(oRs, True, 1, 1, 100, 100, "4,5", saxlsExclude)
```

After import, Styles are applied to the newly imported cells.

```
For i = (FirstRow + 1) To (FirstRow + NumRows)
 cells(i, FirstCol    ).Style = NumStyle   ' OrderID
 cells(i, FirstCol + 1).Style = TxtStyle   ' CustomerID
 cells(i, FirstCol + 2).Style = NumStyle   ' EmployeeID
 cells(i, FirstCol + 3).Style = DateStyle  ' OrderDate
 cells(i, FirstCol + 4).Style = DateStyle  ' RequiredDate
 cells(i, FirstCol + 5).Style = DateStyle  ' Shipped Date
 cells(i, FirstCol + 6).Style = NumStyle   ' Shipped Via
 cells(i, FirstCol + 7).Style = CurrencyStyle  ' Freight
 cells(i, FirstCol + 8).Style = TxtStyle   ' Ship Name
 cells(i, FirstCol + 9).Style = TxtStyle   ' Ship Address
 cells(i, FirstCol +10).Style = TxtStyle   ' Ship City
 cells(i, FirstCol +11).Style = TxtStyle   ' Ship Region
 cells(i, FirstCol +12).Style = TxtStyle   ' Ship PostalCode
 cells(i, FirstCol +13).Style = TxtStyle   ' Ship Country
 cells(i, FirstCol +14).Style = TxtStyle   ' Ship Company Name
 cells(i, FirstCol +15).Style = TxtStyle   ' Address
 cells(i, FirstCol +16).Style = TxtStyle   ' City
 cells(i, FirstCol +17).Style = TxtStyle   ' Region
 cells(i, FirstCol +18).Style = TxtStyle   ' Postal Code
 cells(i, FirstCol +19).Style = TxtStyle   ' Country
Next
```

The following sample demonstrates how to import data from a large recordset using Cells.GetDataFromRecordset and split the recordset rows between multiple worksheets in a workbook.

Adding Charts

How to Create a Chart in Script

ExcelWriter's ExcelApplication API currently supports fifteen Microsoft Excel chart types and over seventy sub-types. For chart type and subtype codes, see Chart Codes. For information on ExcelWriter's charting objects, methods, and properties, see Charts, Chart, SeriesCollection, Series, 3DProperties, Axis, ChartFrame, and Chart Codes.

> ⚠ ExcelWriter is not a charting engine and cannot be used to produce static gif or jpeg representations of charts for inclusion in an HTML page.

This section explains how to create a chart in script. Alternatively, create your charts in Excel, and use ExcelApp.Open or ExcelTemplate to open and modify the spreadsheets. ExcelApp.Open and ExcelTemplate preserve all charts, including chart types that ExcelWriter cannot create.
To add a chart to a spreadsheet, set a Charts object representing the set of all charts in a worksheet:

```
Set objCharts = ws.Charts
```

Using the Charts object's Add method, set a Chart object representing your chart:

```
Set Chart = Charts.Add(ChartType,[SubType],[UpperLeftRow],[UpperLeftCol],
 [LowerRightRow],[LowerRightCol])
```

Consult Chart Codes to select *ChartType* (pie, column, etc.) and *SubType*. *UpperLeftRow*, *UpperLeftColumn*, *LowerRightRow*, and *LowerRightColumn* determine the position of the chart in your spreadsheet.

Set a SeriesCollection object representing the set of all data series in your chart:

```
Set objSeriesCollection = objChart.SeriesCollection
```

Using the CategoryData property, specify the range of category (x) axis values. Assign CategoryData as a range of cells such as "A1:A3" or as a sequence of values such as {5,7,9}. Examples:

```
objSeriesCollection.CategoryData = "A1:A3"
objSeriesCollection.CategoryData = "{5,7,9}"
```

Using the SeriesCollection object's Add method, plot a data series or set of data series:

```
objSeriesCollection.Add(Area, , [Vertical])
```

The parameter *Area* specifies the values from which to plot the data series. Area may be a range of cells such as "A1:C3" or a sequence of values such as {5,7,9}. The optional parameter *Vertical* specifies whether to plot the series from a range of cell values by row or by column. When *Vertical* is set to 0, ExcelWriter will plot the series by row. When Vertical is set to 1, ExcelWriter will plot the series by column. The default value of Vertical is 0.

Accessing Data from Multiple Sheets

The syntax for accessing data in multiple sheets is **SheetName!$Column$Row:$Column$Row, SheetName!$Column$Row:$Column$Row** . For example:

```
SeriesCollection3.Add "Sheet1!$A$1:$C$3,Sheet4!$A$1:$C$3"
```

ExcelWriter does not support the following formulations:

- **Sheet1:Sheet2!$A$1:$C$3**
- **Sheet1!$A$1:$C$3;Sheet4!$A$1:$C$3**

The following example generates a workbook containing three sheets. The first sheet contains the data from which we will plot two charts. We will plot a 3-dimensional clustered column chart in sheet 2 and a radar chart in sheet 3.

```
'--- Declarations
Dim xlw, cells, ws1, ws2, ws3
Dim r
Dim c
Dim row, column, letterA, cellname
Dim Charts_ws2, Chart2, SeriesCollection2
Dim Charts_ws3, Chart3, SeriesCollection3

Set xlw = Server.CreateObject("Softartisans.ExcelWriter")

'--- Create three worksheets in the workbook
Set ws1 = xlw.Worksheets(1)
Set ws2 = xlw.Worksheets(2)
Set ws3 = xlw.Worksheets(3)

'--- Populate cells A1:C3 in the first worksheet
r = 1
c = 1
Do While r < 4
     Do While c < 4
          ws1.Cells.Item(r, c).Value = r + c
          c = c + 1
     Loop
     r = r + 1
     c = 1
Loop

ws1.Cells("A5").Value = "Select Sheet2 to see the chart"

'--- Set a Charts object, representing the set of all charts
'--- in the second worksheet
Set Charts_ws2 = ws2.Charts

'--- In the second sheet, create a 3D clustered column chart
'--- with the upper left corner at row 3, column 0 (A), and
'--- the lower right corner at row 15, column 7 (G).
Set Chart2 = Charts_ws2.Add(0, 3, 3, 0, 15, 7)

'--- In the second sheet, plot a SeriesCollection of three
'--- series, from the values in sheet 1, cells A1:C3.
Set SeriesCollection2 = Chart2.SeriesCollection
SeriesCollection2.CategoryData = "Sheet1!$A$1:$A$3"
SeriesCollection2.Add "Sheet1!$A$1:$C$3"

'--- Set a Charts object, representing the set of all charts in
'--- the third worksheet.
Set Charts_ws3 = ws3.Charts

'--- In the third sheet, create a pie chart with the upper left
'--- corner at row 3, column 0 (A), and the lower right corner
'--- at row 15, column 7 (G).
Set Chart3 = Charts_ws3.Add(4, 0, 3, 0, 15, 7)

'--- In the third sheet, plot a SeriesCollection of three series,
'--- from the values in sheet 1, cells a1 through c3.
Set SeriesCollection3 = Chart3.SeriesCollection
SeriesCollection3.Add "Sheet1!$A$1:$C$3"

'--- Stream the workbook to the client.
xlw.Save "chart.xls", saOpenInPlace

Set xlw = Nothing
Response.End
```

Creating a Complex Chart

When you generate a spreadsheet from a template with the ExcelTemplate object, you can create charts in Excel and the charts will be preserved in the generated spreadsheet. You can use template data markers as source data for the chart and Excel will automatically adjust charts according to the values ExcelWriter assigned to the data marker fields. ExcelTemplate cannot, however, modify chart features at runtime.

With the ExcelApplication object you can generate a chart from script alone and modify every aspect of the chart at runtime. ExcelApplication supports 15 chart types and their sub-types.

The example below shows how to create a complex chart with the ExcelApplication object. The sample demonstrates only a few of the many chart features that you can script with ExcelWriter. You will find all the available chart objects, methods, and properties in the following object reference pages:

- Area
- Axis
- Chart
- ChartFrame
- Charts
- Line
- Series
- SeriesCollection
- 3DProperties

To add a chart to a spreadsheet, use the method Charts.Add. Pass the method a chart type and sub-type and the location of the chart in the specified worksheet – first row, first column, last row, and last column.

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set XlwChart = XlwApp.Worksheets(2).Charts.Add(ChartType, ChartSubType, _
  ChartFirstRow, ChartFirstCol, _
  ChartLastRow, ChartLastCol)
```

To plot a data series in a chart, use SeriesCollection.Add. This method takes a set of values from which to plot the data series, a null second parameter, and an optional boolean parameter that specifies whether to plot the series by row or by column. The method returns a Series object that represents the data series.

```
'--- Add three data series to the chart.
XlwChart.SeriesCollection.Add "Sheet1!B2:B6", , 1
XlwChart.SeriesCollection(1).Name = "March"
XlwChart.SeriesCollection.Add "Sheet1!C2:C6", , 1
XlwChart.SeriesCollection(2).Name = "April"
XlwChart.SeriesCollection.Add "Sheet1!D2:D6", , 1
XlwChart.SeriesCollection(3).Name = "May"
```

Each of the data series added represents a range of Y-axis values. The property SeriesCollection.CategoryData assigns a range of X-axis values from data in the first worksheet:

```
XlwChart.SeriesCollection.CategoryData = "Sheet1!A2:A6"
```

A label can be displayed for each data point in a data series. The labels may show X values, Y values, Y values as percentages, or both X and Y values as percentages. In the example, if the user selected "Actual Value Labels" in the form, labels will display Y values on all charts except pie charts, where labels will display Y values as percentages. If the user selected "Category Labels," labels will display X values on all charts except pie charts, where labels will display X and Y values as percentages.

```
Select Case LabelValues
  Case "ActualValue"
    If ChartType = saxlsPieChart Then
      '--- For a pie chart, show values as percentages of the whole.
      XlwChart.SeriesCollection.LabelShowValueAsPercent = True
    Else
      '--- For non-pie chars, show the actual value above the bars.
      XlwChart.SeriesCollection.LabelShowActualValue = True
    End If
  Case "CategoryLabel"
    If ChartType = saxlsPieChart Then
      '--- For a pie chart, shows percentages and category labels.
      XlwChart.SeriesCollection.LabelShowCategoryAndValueAsPercent = True
    Else
      '--- For a non-pie chart, show the category labels above the bars.
      XlwChart.SeriesCollection.LabelShowCategoryLabel = True
    End If
End Select
```

To display a chart title, set Chart.Title.Text:

```
XlwChart.Title.Text = ChartTitleString
```

A data table is a grid usually attached to the X-axis that displays the source values that the chart is based on. To display a data table, enable the property Chart.ShowDataTable.

```
XlwChart.ShowDataTable = True
```

A ChartFrame object represents an area of the chart. The following properties are ChartFrame objects:ChartArea, DataTable, Legend, LabelFrame, PlotArea, and Title.

```
Set ChartAreaFrame = XlwChart.ChartArea

'--- Stylize the chart area with a shadow (default is False).
ChartAreaFrame.HasShadow = True

'--- Set Pattern = 0 to make it transparent
ChartAreaFrame.Area.Pattern = 0
```

To show a legend, set Chart.ShowLegend to true. To set the legend position within the chart area, set Chart.LegendLocation. Legend items are displayed horizontally by default; to display items vertically, set Chart.LegendVertical to true.

```
'--- Indicate whether or not to show the legend
XlwChart.ShowLegend = True
XlwChart.LegendLocation = saxlsObjectCorner

'--- Should the legend appear vertically or horizontally?
'--- Default is LegendVertical=True.
XlwChart.LegendVertical = False
```

To display an axis title, set Axis.Title.Text.

```
XlwChart.CategoryAxis.Title = CategoryAxisTitle
XlwChart.ValueAxis.Title = ValueAxisTitle
```

See Also

Only a few of the many chart features ExcelWriter offers are demonstrated in this section. For more information, see the following object reference pages:
Area, Axis, Chart, ChartFrame, Charts, Line, Series, SeriesCollection, and 3DProperties.

For information on creating charts with ExcelTemplate, see Templates and Charts.

Creating a PivotTable

A PivotTable is an interactive table that summarizes data to present it in a meaningful way. You can rotate a PivotTable's rows and columns to see different summaries of the source data, or drill down to show details. By displaying different views of data, PivotTable reports allow you to easily compare data, see patterns and relationships, and analyze trends.
In a PivotTable, each column or field in your source data becomes a PivotTable field that summarizes multiple rows of information. In the example below, the Month column becomes the Month field, and each record is summarized in a single item. Sum of Sales is a data field. A data field contains values to be summarized. By default, number values in a data field are summarized by the Sum function and text values are summarized by the Count function. (To change the summarization type set SummarizeBy.) If a PivotTable contains more than one data field, a single field drop-down named Data appears in the report for access to all of the data fields.

Source Data

Product	Sales	Month
A	110	1-Jan-03
A	120	1-Feb-03
B	700	1-Jan-03
B	680	1-Feb-03
C	650	1-Jan-03
C	700	1-Feb-03

PivotTable

Sum of Sales	Product			
Month	A	B	C	Grand Total
1-Jan-03	110	700	650	1460
1-Feb-03	120	680	700	1500
Grand Total	230	1380	1350	2960

This section explains how to create a PivotTable in script. Alternatively, create a PivotTable in Excel and use ExcelApp.Open or ExcelTemplate to open and modify the workbook. ExcelApp.Open and ExcelTemplate preserve existing PivotTables.

Example

The following example opens the worksheet PivotData.xls, which contains source data, and creates a PivotTable based on the source data. Pivotdata.xls contains a named range called DataNamedRange. The source data is in DataNamedRange.

```
'--- Errors will be handled manually in the code
'--- For troubleshooting, comment out this line
'--- to see the default ASP error output
On Error Resume Next

'--- Declarations
Dim XlwApp' As SAExcelApplication
Dim wsPivot' As SAWorksheet
Dim wsData' As SAWorksheet
Dim rngData' As SARange
Dim fldProduct' As IPivotField
Dim fldSales' As IPivotField
Dim fldMonth' As IPivotField
Dim pvtTable' As IPivotTable


'--- Create an instance of SoftArtisans ExcelWriter
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
CheckError Err, "SoftArtisans ExcelWriter could not be created."

'--- Open the workbook with the data we will make into a pivot table
XlwApp.Open("C:\reports\pivotdata.xls")
CheckError Err, "The ""pivotdata.xls"" data workbook could not be opened."

'--- Get a reference to the data sheet
Set wsData = XlwApp.Worksheets("DataSheet")

'--- Create a second worksheet and name it PivotSheet
Set wsPivot = XlwApp.Worksheets(2)
wsPivot.Name = "PivotSheet"

'--- The data in the workbook is in a named range called DataNamedRange
'--- Get a SARange object representing that range
```

```
Set rngData = XlwApp.NamedRange("DataNamedRange")
CheckError Err, "Could not find ""DataNamedRange"" named range."

'--- Every SAWorksheet has a PivotTables collection
'--- Pass a SARange object containing the data for
'--- the pivot table
'--- The range must be a contiguous region of data with
'--- no skipped columns.  The first row of the range must contain
'--- column headers.  All columns must begin on the same row.
Set pvtTable = wsPivot.PivotTables.CreatePivotTable(rngData, 2, 2)
CheckError Err, "Failed to create pivot table."

'--- Properties of IPivotTable
'--- Most properties of IPivotTable are listed here
'--- for demonstration purposes, though they don't all need
'--- to be set explicitly in your code
With pvtTable
  '--- True by default
  .AutoFormat = True
  .DataCollectionInRow = True
  .EnableDrillDown = True
  .EnableFieldDialog = True
  .EnableWizard = True
  .GrandTotalsForColumns = True
  .GrandTotalsForRows = True
  .PreserveFormatting = True
  .RepeatItemLabels = False
  .ShowValueForEmptyCells = True

  '--- False by default
  .ShowFriendlyErrorValue = False
  .InOutlineForm = False
  .MergeLabels = False ' Center label horizontally and vertically
  .SetPrintTitles = False
  .SubtotalHiddenPageItems = False

  .AutoFormatType = saxlAutoFormatReport1
  .Name = "Sales Report 2003" ' Programmatic name for the report

End With

'--- The Fields collection is populated automatically
'--- based on the range of data passed to the CretePivotTable method
Set fldProduct = pvtTable.Fields("Product")
Set fldSales = pvtTable.Fields("Sales")
Set fldMonth = pvtTable.Fields("Month")

'--- Month will be displayed on the left column (Y-axis)
fldMonth.Type = saxlPivotFieldRow

'--- Product name will be displayed across the top (X-axis)
fldProduct.Type = saxlPivotFieldColumn

'--- Sales will be displayed in the body of the table
fldSales.Type = saxlPivotFieldData
'--- Sum up all sales numbers for the table values
fldSales.SummarizeBy = saxlSummarizationSum

'--- Make the pivot table worksheet the active sheet
'--- when the workbook opens
wsPivot.SelectSheet

'--- Stream the workbook to the client
'--- Use saOpenInPlace or saDefault
'--- Pivot tables will not open properly with saOpenInExcel
XlwApp.Save "BasicPivotReport.xls", saOpenInPlace
CheckError Err, "An error occurred saving the workbook."

'--- Dereference ExcelWriter
Set XlwApp = Nothing
```

```
Response.End

'--- This subroutine is called after every major operation
'--- to see if an error has occurred.
'--- If an error has occurred, display a message and stop the script
Sub CheckError(ByRef ThisErr, ByRef msg)
  If ThisErr.number <> 0 Then
    Response.Clear
    Response.Write "<b>" & msg &"</b><br>" & _
        ThisErr.Description & " (" & ThisErr.Source & ")"
```

```
    Response.End
  End If
End Sub
```

The PivotTable object represents a single PivotTable in a worksheet. To create a PivotTable object, use the CreatePivotTable method. Pass CreatePivotTable three parameters:

aDataRange	A Range object containing the source data for the PivotTable. The range must be a contiguous region of data with no skipped columns. The first row of the range must contain column headers. All columns must begin at the same row.
aRow	aRow and aColumnspecify where to put the PivotTable (the table's upper left corner) in the specified worksheet.
aColumn	

In addition to the IPivotTable object, ExcelWriter includes:

- PivotTables
  Represents the set of PivotTables in a worksheet. Use the property Worksheet.PivotTables to access the PivotTables collection.
- PivotFields
  Represents the collection of pivot fields in a PivotTable. A pivot field is a category of data that is derived from a field in the PivotTable's source data. Use the property PivotTable.Fields to access the PivotFields collection.
- PivotField
  Represents a single pivot field in a PivotTable. A pivot field is a category of data that is derived from a field in the PivotTable's source data. To create a PivotField, use the property PivotTable.Fields.

ExcelWriter provides numerous methods and properties that allow you to customize your PivotTable. For more information, see PivotTable and PivotField in the Programmer's Reference.

The Range Object

What is a Range object?

The Range object represents a range of cells within a worksheet. A Range object may include non-adjacent areas. To create a Range object, use, Cells.Range, Cells.Rows, or Cells.Columns.

The property Cells.Range retrieves specified cells into a range object. By default, Cells.Range retrieves all cells in a Cells object. To specify a range of cells within the Cells object, set the Range property's four optional parameters. For example, to set a Range of from row 3 to row 4, and column 10 to column 14, use

```
Set Rng = Cells.Range(3, 10, 2, 5)
```

Range.Cells Optional Parameter	Default Value	
FirstRow	1	
FirstCol	1	
NumRows	65536	ExcelWriter Limit imposed by Microsoft's BIFF8 format
500	ExcelWriterFree Edition Limit	
NumCols	256	Limit imposed by Microsoft's BIFF8 format, not by ExcelWriter

- Example 1: Outlines
- Example 2: Union, Intersection, Names, and Outlines
- Example 3: Using Range to Set PrintArea

Example 1: Outlines

The following example uses GroupColumns and GroupRows to create outlines within a range.

The first row or column in an outline is the summary row or column, and is always visible. The rows or columns following the summary are the outline detail rows or columns, and will be visible or hidden, depending on the value of the Collapsed parameter. For definitions of GroupColumns/Rows parameters, see the following table.

```
Dim xlw, ws, rng

Set xlw = Server.CreateObject("Softartisans.Excelwriter")
Set ws = xlw.Worksheets(1)
Set rng = ws.Cells.Range(1, 1, 20, 10)

ws.Cells("A2").Value = "Product2"
ws.Cells("A3").Value = "Product3"
ws.Cells("A4").Value = "Product4"
ws.Cells("A5").Value = "Product5"
ws.Cells("A6").Value = "Product6"
ws.Cells("A7").Value = "Product7"
ws.Cells("A8").Value = "Product8"
ws.Cells("A9").Value = "Product9"
ws.Cells("A10").Value = "Product10"
ws.Cells("A11").Value = "Product11"
ws.Cells("A12").Value = "Product12"
ws.Cells("A13").Value = "Product13"
ws.Cells("A2").Value = "Product14"
ws.Cells("B3").Value = "Product15"
ws.Cells("C4").Value = "Product16"
ws.Cells("D5").Value = "Product17"
ws.Cells("E6").Value = "Product18"
ws.Cells("F7").Value = "Product19"
ws.Cells("G8").Value = "Product20"
ws.Cells("H9").Value = "Product21"
ws.Cells("I10").Value = "Product22"
ws.Cells("J11").Value = "Product23"
ws.Cells("K12").Value = "Product24"
ws.Cells("L13").Value = "Product25"

rng.GroupRows 4, 4, True
rng.GroupRows 9, 5, False
rng.GroupRows 3, 11, True
rng.GroupColumns 5, 3, False
rng.GroupColumns 4, 5, True
rng.GroupColumns 3, 7, False
rng.GroupColumns 2, 9, True

xlw.Save "Filename.xls", saOpenInPlace
Response.End
Set xlw = Nothing
```

Example 2: Union, Intersection, Names, and Outlines

The following example imports data from the Northwind database to an Excel spreadsheet. The script sets five ranges, and names each using the Name property. The range names will be listed in Excel's drop-down name box above the top left corner of the spreadsheet. Select a range name from the list to highlight a range.

- Range1 is defined as the set of cells beginning at row 4, column 2, and spanning 4 rows and 5 columns.
- Later Range1 is redefined using the Union method, which creates a union of the two ranges Range1 and Range2.
- Range2 - originally the set of cells from cell (6, 3) to cell (12, 11) - is redefined by the Intersect method as the intersection of Range2 and Range3.
- GroupColumns and GroupRows are used to create outlines within Range4 and Range5. The first row or column in an outline is the summary row or column, and is always visible. The rows or columns following the summary are the outline detail rows or columns, and will be visible or hidden, depending on the value of the Collapsed parameter. For definitions of GroupColumns/Rows parameters, see the table below. The following line outlines columns 13 to 20, and hides the outline detail columns.

```
Dim filePath
Dim oConn, oRs, xlw, ws1, FirstRow, FirstCol
Dim AllRange, Range1, Range2, Range3, Range4, Range5, Range6
Dim DateStyle, ScriptName
```

```
Set oConn = Server.CreateObject("ADODB.Connection")
oConn.Open Application("connstring")
Set oRs = oConn.Execute ("select * from Orders where ShipRegion='AK' " & _
     "or ShipRegion='WA' or ShipRegion='NM' order by ShipRegion" )

Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set ws1 = xlw.Worksheets(1)

'--- Convert Recordset output to workbook input
FirstRow = 3
FirstCol = 2

'--- Import the data in one step
'--- We add 1 to FirstRow to leave room for the headers
Set AllRange = ws1.Cells.GetDataFromRecordset(oRs, True, FirstRow, FirstCol)

'--- Define Range1 beginning at row 4, column 2, and
'--- spanning 4 rows and 5 columns. Name the range
'--- "Range1."
Set Range1 = ws1.Cells.Range(4, 2, 4, 5)
Range1.Name = "Range1"

'--- Define Range2 beginning at row 6, column 3, and
'--- spanning 7 rows and 9 columns. Name the range
'--- "Range2."
Set Range2 = ws1.Cells.Range(6, 3, 7, 9)
Range2.Name = "Range2"

'--- Define Range3 beginning at row 3, column 9, and
'--- spanning 5 rows and 12 columns.  Name the range
'--- "Range3."
Set Range3 = ws1.Cells.Range(3, 9, 5, 12)
Range3.Name = "Range3"

'--- Expand Range1 by creating a union between Range1
'--- and Range2.
Range1.Union(Range2)

'--- Redefine Range2 as the intersection of Range2 and
'--- Range3.
Range2.Intersect(Range3)

'--- Define Range4 beginning at row 14, column 2, and
'--- spanning 18 rows and 20 columns.  Name the range
'--- "Range4."
Set Range4 = ws1.Cells.Range(14, 2, 18, 20)
Range4.Name = "Range4"

'--- Outline rows 14 to 31, and hide the outline detail
'--- rows by setting the Collapsed parameter to True.
'--- Row 14 is the summary row and will be visible.
Range4.GroupRows 1, 17, True

'--- Outline columns 9 to 12, and hide the outline detail
'--- columns by setting the Collapsed parameter to True.
'--- Column 9 is the summary column and will be visible.
Range4.GroupColumns 9, 2, True

'--- Outline columns 13 to 20, and hide the outline detail
'--- columns by setting the Collapsed parameter to True.
'--- Column 13 is the summary column and will be visible.
Range4.GroupColumns 13, 7, True

'--- Define Range5 beginning at row 33, column 2, and
'--- spanning 19 rows and 20 columns.  Name the range
'--- "Range5."
Set Range5 = ws1.Cells.Range(32, 2, 19, 20)
Range5.Name = "Range5"

'--- Outline rows 33 to 50, and show the outline detail rows
```

```
'--- by setting the Collapsed parameter to False (which is
'--- the default value).  Row 33 is the summary row.  Rows 34
'--- to 50 are the outline detail rows.
Range5.GroupRows 1, 18, false

'--- Define a style that will format the cells into a date format.
'--- Apply the style on a range that references row 4 to 50 and
'--- column 4 through 5(E,F and G).
Set DateStyle = xlw.CreateStyle
DateStyle.Number = 14
DateStyle.WrapText = True

Set Range6 =  ws1.Cells.Range(4, 5, 46, 3)
Range6.Style = DateStyle
Range6.ColumnWidth = 20

'--- Open the spreadsheet in the browser.
xlw.Save "Range1.xls", saOpenInPlace

'--- Error handling.
ScriptName = Request.ServerVariables("Script_Name")
If Err.number <> 0 Then
  Response.Status = "500 Internal Server Error"
  Response.Write "<P><H2>Creation of Excel Spreadsheet Failed.<BR>" &_
    ScriptName & "</H2><BR>"
  Response.Write "Error : '" & Err.description & "'</P>"
Else
  Response.End
End If
```

```
Response.End

Set xlw = Nothing
```

Example 3: Use Range to Set PrintArea

The Range object can be used to assign a certain group of cells to ExcelWriter's PrintArea property. A PrintArea is a Range that is read by the PageSetup object when determining the portion of the page that should be printed.

A Range can be directly assigned to a PrintArea:

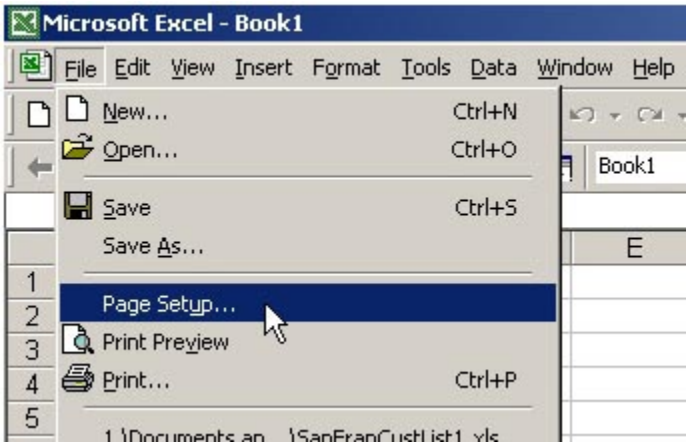
```
Ws.PageSetup.PrintArea = Ws.Cells.Range(1, 1, 20, 10)
```

Or it can be assigned to a variable which can be used to set the PrintArea:

```
Set pa = Ws.Cells.Range(1, 1, 20, 10)
Ws.PageSetup.PrintArea = pa
```

Page Setup

In Microsoft Excel, you can set the layout properties of a printed worksheet by selecting **Page Setup...** from the **File** menu:



If you set page setup properties in Microsoft Excel and open the spreadsheet with ExcelWriter, the page setup settings will be preserved. Alternatively, use ExcelWriter's PageSetup object to set layout properties in script.

ExcelWriter's PageSetup object represents the layout properties of a printed worksheet. To create a PageSetup object with ExcelWriter, use:

```
Set PageSetup = Worksheet.PageSetup
```

> ⚠ 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.

Example

```
Dim xlw, ws, commentstyle, ScriptName
Dim NumRows, NumCols, NumSheets
Dim RowCount, ColCount, SheetCount
Dim SRowX, SColY, SSheetZ
```

```
NumRows = 40
NumCols = 40
NumSheets = 1

RowCount = NumRows + 1
ColCount = NumCols + 1
SheetCount = NumSheets
SRowX = 3
SColY = 1
SSheetZ = 1

Set xlw =  Server.CreateObject("SoftArtisans.ExcelWriter")
Set ws = xlw.Worksheets(1)

'--- Set a comment style
Set commentstyle = xlw.CreateStyle
commentstyle.Font.Name = "Arial"
commentstyle.Font.Bold = True
commentstyle.VerticalAlignment = savaCenter
commentstyle.HorizontalAlignment = sahaCenter

'--- Include a comment explaining where to
'--- view the page setup settings
ws.Cells.MergeCells 1,1,1,15
ws.Cells.RowHeight(1) = 20
ws.Cells("A1").Style = commentstyle
ws.Cells("A1").Value = "PageSetup.asp assigned layout settings for " &_
     "printing this spreadsheet.  To see the layout " &_
     "settings, open File/Page Setup..." &_
     "and File/Print Preview."

'--- Set layout properties for the printed
'--- worksheet.
ws.PageSetup.CenterFooter = "centerfooter"
ws.PageSetup.CenterHeader = "centerheader"
ws.PageSetup.CenterVertically = True
ws.PageSetup.CenterHorizontally = True
ws.PageSetup.LeftFooter = "leftfooter"
ws.PageSetup.RightFooter = "rightfooter"
ws.PageSetup.LeftHeader = "leftheader"
ws.PageSetup.RightHeader = "rightheader"
ws.PageSetup.Orientation = 1
ws.PageSetup.PaperSize = saxlPaperA4
ws.PageSetup.FirstPageNumber = 2
ws.PageSetup.Zoom = 50
Ws.PageSetup.FitToPagesTall = 2
Ws.PageSetup.FitToPagesWide = 3

'--- Set title columns and rows that will be repeated on each page
'--- of a printed worksheet.  SetPrintTitleColumns and
'--- SetPrintTitleRows set two Areas of the Range returned
'--- by PageSetup.GetPrintTitles.
Ws.PageSetup.SetPrintTitleRows 4, 1
Ws.PageSetup.SetPrintTitleColumns 4, 1

'--- Define theworksheets print area.  PrintArea is a Range
'--- that is read by the PageSetup object when determining the
'--- portion of the page that should be printed.
Ws.PageSetup.PrintArea = Ws.Cells.Range(20, 20, 10, 10)

Do While SRowX < RowCount
     Do While SColY < ColCount
          xlw.Worksheets(1).Cells(SRowX, SColY).Value = "PageSetup"
          SColY = SColY + 1
     Loop
     SColY = 1
     SRowX = SRowX + 1
Loop

'--- Save the spreadsheet.
```

```
xlw.Save "PageSetup.xls", saOpenInPlace

'--- Error handling.
ScriptName = Request.ServerVariables("Script_Name")
If Err.number <> 0 Then
     Response.Status = "500 Internal Server Error"
     Response.Write "Error : '" & Err.description & "'</P>"
Else
     Response.End
```

```
End If

Set xlw = Nothing
```

Formatting Headers and Footers

Formatting Headers and Footers with ExcelWriter

ExcelWriter allows you to create and format headers and footers for your spreadsheet in script using the the PageSetup object properties LeftFooter, CenterFooter, RightFooter, LeftHeader, CenterHeader, and RightHeader.

To format a header or footer in script, include formatting codes within the ExcelWriter header/footer property. For example, the following displays the center header text in bold Tahoma:

```
ws.PageSetUp.CenterHeader = "&""Tahoma""&BFormatting Headers and Footers&B"
```

The following example demonstrates formatting headers and footers in an ExcelWriter script.

```
Dim oXLW
Dim oWS

Set oXLW = Server.CreateObject("Softartisans.ExcelWriter")
Set oWS = oXLW.Worksheets(1)

oWS.Cells(2, 1).Value = "To see the headers and footers, open the File menu, "
oWS.Cells(3, 1).Value = "select Page Setup, and select Header/Footer."

'---Format the headers and footers using VBA format codes
oWS.PageSetUp.LeftHeader = "&""Tahoma, Bold""&16&ESoftArtisans&E"
oWS.PageSetUp.CenterHeader = "&""Tahoma""&IFormatting Headers/Footers&I"
oWS.PageSetUp.RightHeader = "&""Tahoma""&A"
oWS.PageSetUp.LeftFooter = "&""Tahoma""&D"
oWS.PageSetUp.CenterFooter = "&""Tahoma""&T"
oWS.PageSetUp.RightFooter = "&""Tahoma""Page &P"

oXLW.Save "out.xls", saOpenInPlace
Set oXLW = Nothing
```

Formatting Codes for Headers and Footers	
&L	Left aligns the characters that follow.
&C	Centers the characters that follow.
&R	Right aligns the characters that follow.
&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.

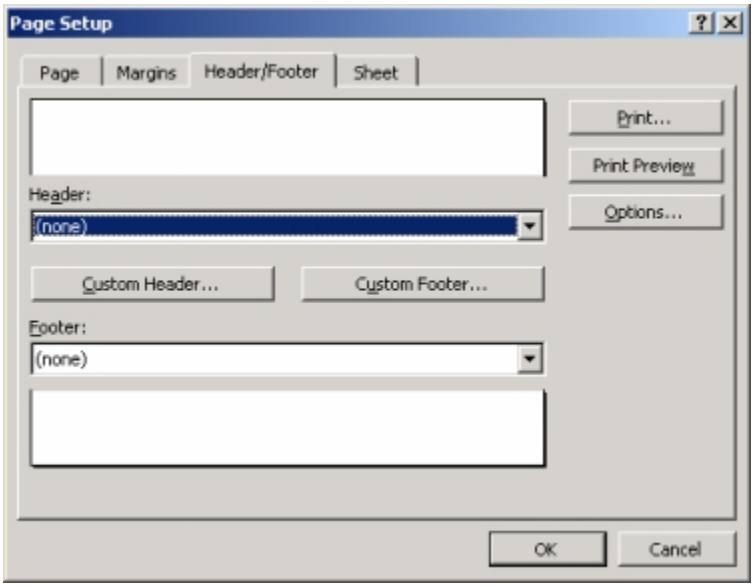
&F	Prints the name of the document.
&A	Prints the name of the worksheet.
&P	Prints the page number.
&P+*number*	Prints the page number plus the specified number.
&P-*number*	Prints the page number minus the specified number.
&&	Prints a single ampersand.
&"*font name*"	Prints the characters that follow in the specified font size. In ASP, the font name must be within double quotation marks, as in the following example, `ws.PageSetUp.CenterHeader = "&""Tahoma""&Center Header"`
&*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

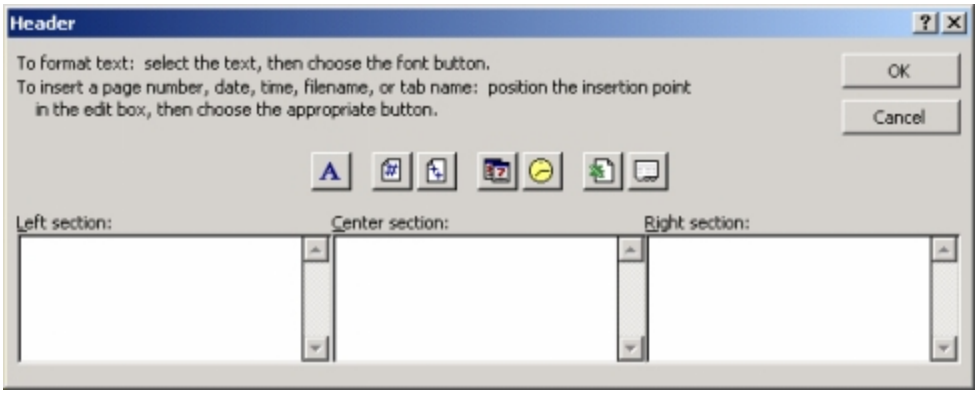
When opening an existing Excel spreadsheet, ExcelWriter will preserve the header and footer formatting created in Excel. However, if you change the header/footer text in your ExcelWriter script and do not include Formatting Codes, the formatting created in Excel will be overwritten and default formatting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

Protecting Your Worksheet

Write Protection, Not Encryption

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. The ability to present your spreadsheet to end users in a visible, but uneditable format can be very useful for presentation purposes. You can implement worksheet protection using ExcelWriter by setting the WriteProtect property of the Worksheet object.

It is important to note that 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. Furthermore, any workbook with worksheet protection can be opened and modified using ExcelWriter.

- How to Activate Worksheet Protection with ExcelWriter
- Removing Worksheet Protection
- Protecting All Worksheets in a Workbook
- Protecting Specific Cells
- Example: Generating a Workbook with Worksheet Protection

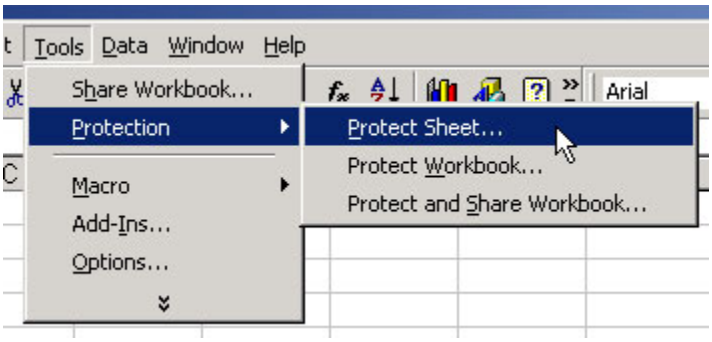
How to Activate Worksheet Protection with ExcelWriter

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 an ExcelWriter Script

- **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 in an ExcelWriter script using either ExcelTemplate or ExcelApplication.Open. ExcelWriter will preserve all existing spreadsheet settings, including worksheet protection.



- **Protect in an ExcelWriter Script**
  Use the property Worksheet.WriteProtect to activate worksheet protection in the ExcelWriter script as follows:

  1. **Optional:** Determine your **passcode**. If you do not want to set a password to disable write protection, continue to step 2. In ExcelWriter, the passcode is an integer value that corresponds to a clear-text password string. Excel's password hashing algorithm is private, as such it is not possible to set Worksheet.WriteProtect to a regular string. For example, setting WriteProtect to -1843 sets the worksheet password to "mypassword".

     To determine which passcode correponds to a given password string, create a blank workbook in Excel and supply your desired password in the Worksheet Protection creation process. Then, open the workbook using ExcelApplication.Open and read the value of WriteProtect. For example, if your workbook is called "password.xls", use this code to determine the *passcode* that correponds to the password string you've set for the first worksheet:

     ```
     Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
     xlw.Open "password.xls"
     Response.Write(xlw.Worksheets(1).WriteProtect)
     ```

  2. Set WriteProtect to your passcode, for example:

     ```
     Worksheet.WriteProtect = -1843
     ```

     OR, to enable write protection without setting a password:

     ```
     Worksheet.WriteProtect = True
     ```

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 in an ExcelWriter script, set Worksheet.WriteProtect to False:

```
Worksheet.WriteProtect = False
```

Protecting All Worksheets in a Workbook

ExcelWriter applies write protection on a per-worksheet basis. If you want to apply protection to all of the worksheets in your workbook, you will need to set it for each worksheet individually. You can use a simple loop like this:

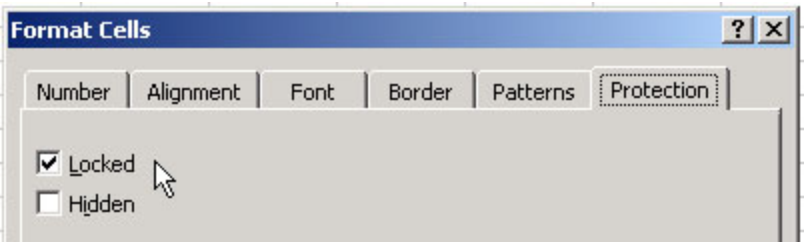
```
For Each worksheet In xlw.Worksheets
     worksheet.WriteProtect = passcode
Next
```

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 a style's Locked property to False and applying the style to an individual cell. The following example protects all cells except for A1 and B1:

```
'--- Turn on worksheet protection
WorkSheet.WriteProtect = -1843

'--- Create an "unlocked" style
Set unlockedStyle = xlw.CreateStyle
unlockedStyle.Locked = False

'--- Apply the style
Worksheet.Cells("A1").Style = unlockedStyle
Worksheet.Cells("B1").Style = unlockedStyle
```

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
- 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
```

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)
```

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.

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 for ASP.NET or 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.

See also, Passing ExcelTemplate to ExcelApplication.

To create a template, use the ExcelApplication object to create a workbook and insert data markers in cells:

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
CheckError Err, "SoftArtisans ExcelWriter could not be created."

'--- Create a worksheet and get a reference to it.
Set WrkSht = XlwApp.Worksheets(1)

'--- Loop through the selected columns.
For i = 1 To ColumnCount
  '--- Form a data marker to be inserted into the workbook
  '--- Data marker syntax: %%=[DataSrcName].[ColName](<modifier>)
  CurMarker = "%%=" & DataSrcName & "." & Columns(i - 1)

  '--- Insert (fieldname) data markers in the first row.
  '--- Stylize the header row with bold faced font.
  WrkSht.Cells(FirstRow, FirstCol + (i - 1)).Value = CurMarker & "(fieldname)"
  WrkSht.Cells(FirstRow, FirstCol + (i - 1)).Format.Font.Bold = True

  '--- And push down the data markers one row to make room for the headers
  WrkSht.Cells(FirstRow + 1, FirstCol + (i - 1)).Value = CurMarker
Next
```

A data marker will be created dynamically for each database column. A data marker is a cell value beginning with **%%=** or **%%=$** that specifies a variable, an array, or, as in this cass, a database column, to insert in the spreadsheet column containing the marker. The data marker may be followed by a modifier. To include column headers in the populated data marker column, the data marker should include the (fieldname) modifier (for example, **%%=DataSource.EmployeeID(fieldname)**).

To return the workbook created with ExcelApplication as an ExcelTemplate object, pass saOpenAsTemplate to ExcelApplication's Save method.

```
Set XlwTempl = XlwApp.Save("", saOpenAsTemplate)
CheckError Err, "Error opening the workbook as ExcelTemplate."
```

Next ExcelTemplate will populate the template data markers with records retrieved from the Employees table in the Northwind database (Northwind.mdb), and stream the generated spreadsheet to the client:

```
'--- Create and open an ADODB Connection.
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("connstring")
CheckError Err, "An error occurred opening the database connection."

'--- Get a RecordSet of customer information
Set RecSet = Conn.Execute("SELECT " & ColumnList & " FROM Employees")
CheckError Err, "An error occurred querying the database."

'--- Bind the RecordSet to the data markers that were inserted
'--- with ExcelApplication.
XlwTempl.DataSource(DataSrcName) = RecSet

'--- Process and stream the populated workbook to the client.
XlwTempl.Process WorkbookName, saProcessOpenInExcel, True
CheckError Err, "An error occurred processing the template."
```

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 machine. HotCell Technology uses Excel VBA code embedded in the spreadsheet on the client. The VBA code communicates with an ASP.NET or ASP script running on a Web server. The VBA code can be executed when the user modifies a cell's value. A "HotCell" is a cell in the client-side spreadsheet that is modified by the user.

Updating a Server-Side Data Source with HotCell Technology

HotCell Technology allows you to update a server-side data source based on changes a user made to a client-side Excel spreadsheet that contains HotCells. This solution utilizes client-side Excel VBA code to detect when changes have been made to worksheet cells. The address and

value of each modified cell is submitted to the server in order to modify the data source. There are two different upload methods that you can use to submit the changed cell data back to the server:

1. **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 OWAssist.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, see the OfficeWriter Assistant documentation.

2. **POST changed-cell instructions to the server**

   This method submits cell information in an HTTP POST request. On the server, the receiving ASP.NET or ASP script 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.

POST Example: Editing Cells with Known Positions

The POST example contains the following files:

- Form.asp

  Form.asp is the sample's entry point. The form contains a drop-down list of employee IDs. The user will select an ID and click the "Get Employee Data Form" button to open an employee data-update form (an Excel spreadsheet). The button also opens a browser window that displays the employee's database record as it appears in the Employees database table.

- GetEmployeeDataSheet.asp

  This script is activated when you click Form.asp's "Get Employee Data Form" button. The script gets the selected employee's database record, opens an ExcelWriter template, populates the template with the employee's data, and generates an Excel spreadsheet. The generated spreadsheet displays employee data and allows you to modify the data and submit the changes to update the database. When you click "Update" to submit the changes, VBA code uses the XMLHttp client object to send HotCell name/value pairs to Update.asp.

- Update.asp

  Update.asp takes the new values sent from GetEmployeeDataSheet.asp and updates the employee's database record.

- DatabaseView.asp

  DatabaseView.asp opens a browser window that displays the selected employee's database record in a table. The database view window opens when you click "Get Employee Data Form" before any modifications are made. After you modify the employee data and the database is updated, refresh the database view window to see the changes.

- BasicFormUpdateTemplate.xls

  Template Excel workbook.

To see the VBA code go to Tools > Macros > Visual Basic Editor.

Form.asp

```
<%@ Language="VBScript" %>
<% Option Explicit %>
<%
'--- ErrRecSet will be handled manually in the code
'--- For troubleshooting, comment out this line
'--- to see the default ASP error output
On Error Resume Next

'--- Declarations
Dim Conn, RecSet

'--- Instantiate and open a database connection
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("connstring")

'--- Query the database to get a list of employee ids
Set RecSet = Conn.Execute("SELECT EmployeeID, (LastName+', '+FirstName) _
 As Name FROM Employees ORDER BY EmployeeID")
%>
<HTML>
<HEAD>
<OBJECT
classid="clsid:381A579F-C095-433d-8AEF-96CED6EC227F"
codebase="OWAssist.cab" ID="Object1">
</OBJECT>
</HEAD>
<BODY topmargin="0" leftmargin="0" marginwidth="0" marginheight="0" bottommargin="0">
<P>This application allows you to edit data about employees
of the SoftArtisans Traders Company through an online Excel
workbook. Select an Employee ID for the employee whose information
you wish to edit.</P>
<p><b>Note:</b> When you click the button, a new browser window will
open to allow you to monitor the changes that you are submitting
to the database with the HotCell-enabled workbook. </p>
<BR><BR>
<FORM ACTION="GetEmployeeDataSheet.asp" METHOD="POST">
<SELECT NAME="EmployeeID">
<%
'--- For each employee ID, write a option for our drop-down box
Do While Not RecSet.EOF
  Response.Write "<OPTION VALUE="""" & RecSet.Fields("EmployeeID").Value & _
  """>" & RecSet.Fields("Name") & "</OPTION>"
  RecSet.MoveNext
Loop

'--- Clean up
RecSet.Close
Conn.Close
Set RecSet = Nothing
Set Conn = Nothing
%>
</SELECT>
<BR><BR>
<INPUT TYPE="submit" onClick="javascript:window.open('DatabaseView.asp');"
VALUE="Get Employee Data Form">
</FORM>
</BODY>
</HTML>
```

GetEmployeeDataSheet.asp

```
<%@ Language="VBScript" %>
<% Option Explicit %>
<!--METADATA TYPE="TypeLib" UUID="{7BCD2133-64A0-4770-843C-090637114583}" -->
```

```
<%
On Error Resume Next

'--- Declarations
Dim Conn
Dim RecSet
Dim XlwTemplate
Dim EmployeeID
Dim SQL

'--- Get the submitted Employee ID from the form
EmployeeID = Request.Form("EmployeeID")

SQL = "SELECT EmployeeID, FirstName, LastName, Title, TitleOfCourtesy, " & _
"BirthDate, City, Country, Region, Address, PostalCode, HomePhone, " & _
"Extension, Notes FROM Employees WHERE EmployeeID=" & EmployeeID

'--- Open a database connection
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("connstring")
CheckError Err, "The database connection could not be opened."

'--- Query the database to get information about the chosen employee
Set RecSet = Conn.Execute(SQL)
CheckError Err, "The database query has failed."

'--- Instantiate the ExcelTemplate object and open the template workbook
Set XlwTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")
XlwTemplate.Open Server.MapPath("./BasicFormUpdateTemplate.xls")
CheckError Err, "ExcelTemplate could not open the template workbook."

'--- Tell Excel to preserve strings that are numeric by nature as strings.
'--- This will avoid losing leading "0"s from data such as zip codes.
XlwTemplate.PreserveStrings = True

'--- Set the ExcelTemplate datasource to the recordset
XlwTemplate.DataSource("Emp") = RecSet

'--- Set the postURL in a hidden cell in the workbook
'--- This is the URL to which the HotCell workbook should post
'--- database updates.  Use this technique to be sure that the
'--- HotCell workbook always posts back to the server on which
'--- it was generated
XlwTemplate.DataSource("HotCellPostUrl") = "http://" &_
 Request.ServerVariables("SERVER_NAME") & _
 Application("vroot") & _
 "/HotCells/BasicFormUpdate/ASP/update.asp"

'--- Save and stream the populated workbook
'--- "EmployeeForm.xls" name that will appear in Save As dialog
'--- "saProcessOpenInExcel" option to open the workbook in a new instance of Excel
'--- "False" exclude macros - HotCell templates require that VBA macros be enabled
XlwTemplate.Process "EmployeeForm.xls", saProcessOpenInExcel, False
CheckError Err, "The template could not be processed."

'--- Close ExcelTemplate
XlwTemplate.Close : Set XlwTemplate = Nothing

'--- Close ADO objects
RecSet.Close : Set RecSet = Nothing
Conn.Close : Set Conn = Nothing

'--- This subroutine is called after every major operation
'--- to see if an error has occurred.
'--- If an error has occurred, display a message and stop the script
Sub CheckError(ByRef ThisErr, ByRef msg)
  If ThisErr.number <> 0 Then
    Response.Clear
    Response.Write "<b>" & msg & "</b><br>" & _
      ThisErr.Description & " (" & ThisErr.Source & ")"
```

```
Response.End
```

```
  End If
End Sub
%>
```

Update.asp

```
<%@ Language="VBScript" %>
<% Option Explicit %>
<!-- #include file="adovbs.inc" -->
<%
On Error Resume Next

Dim Conn
Dim oCmd
Dim strSQL
Dim employeeID
Dim strAddress
Dim strCity
Dim strRegion
Dim strPostalCode
Dim strCountry
Dim strHomePhone
Dim strExtension
Dim strNotes
Dim intRowsAffected
Dim p1, p2, p3, p4, p5, p6, p7, p8, p9

'--- Get the values from the form that was submitted by
'--- the HotCell workbook.
With Request
  employeeID = .Form("employeeid")
  strAddress = .Form("address")
  strCity = .Form("city")
  strRegion = .Form("region")
  strPostalCode = .Form("postalcode")
  strCountry = .Form("country")
  strHomePhone = .Form("homephone")
  strExtension = .Form("extension")
  strNotes = .Form("notes")
End With

'--- Form the SQL statement using the values submitted by
'--- the Excel macro, via the Request.Form object.
'--- Create the ADODB.Connection object and open the
'--- connection with the appropriate connection string.
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("connstring")
If Err.number <> 0 Then
  Response.Write Err.Description
  Response.Status = 500
  Response.End
End If

'--- Create an ADO command to handle the update
Set oCmd = Server.CreateObject("ADODB.Command")
oCmd.ActiveConnection = Conn

'--- The command will execute this SQL query
strSQL = "UPDATE Employees SET " &_
  "Address=?, City=?, Region=?, PostalCode=?, " &_
  "Country=?, HomePhone=?, Extension=?, Notes=? " &_
  "WHERE EmployeeID=?"

oCmd.CommandText = strSQL
```

```
Set p1 = oCmd.CreateParameter("@Address",_
 adVarChar, adParamInput, 100, strAddress)
oCmd.Parameters.Append p1
Set p2 = oCmd.CreateParameter("@City",_
    adVarChar, adParamInput, 100, strCity)
oCmd.Parameters.Append p2
Set p3 = oCmd.CreateParameter("@Region",_
    adVarChar, adParamInput, 100, strRegion)
oCmd.Parameters.Append p3
Set p4 = oCmd.CreateParameter("@PostalCode",_
    adVarChar, adParamInput, 100, strPostalCode)
oCmd.Parameters.Append p4
Set p5 = oCmd.CreateParameter("@Country",_
    adVarChar, adParamInput, 100, strCountry)
oCmd.Parameters.Append p5
Set p6 = oCmd.CreateParameter("@HomePhone",_
    adVarChar, adParamInput, 100, strHomePhone)
oCmd.Parameters.Append p6
Set p7 = oCmd.CreateParameter("@Extension",_
    adVarChar, adParamInput, 100, strExtension)
oCmd.Parameters.Append p7
Set p8 = oCmd.CreateParameter("@Notes",_
    adVarChar, adParamInput, 1000, strNotes)
oCmd.Parameters.Append p8
Set p9 = oCmd.CreateParameter("@EmployeeID",_
    adInteger, adParamInput, , employeeID)
oCmd.Parameters.Append p9

'--- Execute the SQL statement we just formed above
If Request.ServerVariables("REQUEST_METHOD") = "POST" Then
  '--- The "128" is an ADO contant that tells the object
  '--- that we're not expecting any recordset back from this
  '--- call to the Execute method
  oCmd.Execute intRowsAffected
  If Err.number <> 0 Then
    Response.Write Err.Description
    Response.Status = 500
    Response.End
  End If
 End If

Conn.Close
Response.End

'--- Use this code to figure out what error occurred, if any.
'--- An error decsription log will be written to the script directory
'--- This log can be very useful in debugging a HotCell application because
'--- it will capture the error that occurred on the server.
'--- The web server needs write access to the log directory in order for
'--- this to function properly
If Err.Number <> 0 Then
  Dim db
  Set db = Server.CreateObject("EZDebug.DebugString")
  db.OutputDebugString strSQL
  db.OutputDebugString "RA: " &intRowsAffected
  db.OutputDebugString Err.number &" : " &Err.Description &_
      " : " &Err.Source
  Set db = Nothing
  Response.Status = 500
End If

'--- Close the connection
Conn.Close
Set Conn = Nothing

'--- This subroutine is called after every major operation
'--- to see if an error has occurred.
'--- If an error has occurred, display a message and stop the script
Sub CheckError(ByRef ThisErr, ByRef msg)
```

```
If ThisErr.number <> 0 Then
  Response.Clear
  Response.Write "<b>" &msg &"</b><br>" &_
      ThisErr.Description &" (" &ThisErr.Source &")"
  Response.End
```

```
  End If
End Sub
%>
```

DatabaseView.asp

```
<html>
<head>
<title>OfficeWriter - Database Viewer</title>
</head>
<body>
<form>
<input type="button" value="Refresh to View Changes"
onclick="javascript:window.location.reload(true);">
</form>
<h3>Database Viewer: Employees table</h3>

<p>Click "Refresh" to see your changes after you submit them from
the HotCell worksheet.</p>

<table border=1>
<%
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("connstring")

 SQL = "SELECT EmployeeID As [Employee ID], FirstName As [First Name], " &_
     LastName as [Last Name], Title, " & _
     "TitleOfCourtesy As [Title Of Courtesy], " & _
     "BirthDate As [Birth Date], City, Country, Region, Address, " & _
     "PostalCode As [Postal Code], HomePhone As [Home Phone], " & _
     "Extension, Notes FROM Employees ORDER BY EmployeeID"

Set RecSet = Conn.Execute(SQL)

Response.Write "<TR>"
For i=0 To RecSet.Fields.Count-1
 Response.Write "<TH>" & RecSet.Fields(i).Name & "</TH>"
Next
Response.Write "</TR>"


Do While Not RecSet.EOF
 Response.Write "<TR>"
 For i=0 To RecSet.Fields.Count-1
  Response.Write "<TD> " & RecSet.Fields(i).Value & "</TD>"
 Next
 Response.Write "</TR>"
 RecSet.MoveNext
Loop
%>
</table>
</body>
</html>
```

Programmer's Reference

- Object Model
- ExcelTemplate Programmer's Reference
  - ExcelTemplate
    - ExcelTemplate.AnsiToUnicode(String, Long)
    - ExcelTemplate.ContentType
    - ExcelTemplate.DataSource
    - ExcelTemplate.DecryptPassword
    - ExcelTemplate.EncryptPassword

        - ExcelTemplate.Open(String)
        - ExcelTemplate.PreserveStrings
        - ExcelTemplate.Process(String, SAProcessMethod, Long)
        - ExcelTemplate.SpreadsheetsCreatedToday
        - ExcelTemplate.UnicodeToAnsi(String, Long)
- ExcelApplication Programmer's Reference
    - 3DProperties
        - 3DProperties.ChartDepth
        - 3DProperties.DistanceFromEyeToChart
        - 3DProperties.ElevationAngle
        - 3DProperties.PlotVolumeHeight
        - 3DProperties.RotationAngle
        - 3DProperties.SeriesGapDepth
        - 3DProperties.Show3D
        - 3DProperties.Stack3DColumns
        - 3DProperties.UseAutoScaling
        - 3DProperties.UsePerspectiveTransform
        - 3DProperties.ZAxis
    - Area
        - Area.AreaFormatting
        - Area.BackgroundColor
        - Area.ForegroundColor
        - Area.Pattern
    - Axis
        - Axis.AxisCrossesMidCategory
        - Axis.AxisLine
        - Axis.CategoryAxisCrossMaxValue
        - Axis.CategoryCrossingValue
        - Axis.LabelFont
        - Axis.LabelPosition
        - Axis.LogarithmicScale
        - Axis.MajorGridline
        - Axis.MajorIncrement
        - Axis.MajorTickType
        - Axis.MaxValue
        - Axis.MinorGridline
        - Axis.MinorIncrement
        - Axis.MinorTickType
        - Axis.MinValue
        - Axis.Number
        - Axis.TextRotationAngle
        - Axis.Title
        - Axis.ValuesInReverseOrder
        - Axis.Wall3D
        - Axis.Wall3DBorder
    - Cell
        - Cell.Characters
        - Cell.ClearContents()
        - Cell.ClearFormats()
        - Cell.Col
        - Cell.Format
        - Cell.Formula
        - Cell.Name
        - Cell.PutValue(Variant)
        - Cell.Row
        - Cell.SetAddInFormula(String, String)
        - Cell.Style
        - Cell.Value
    - Cells
        - Cells.AddHyperlink(Long, Long, Long, Long, String, String())
        - Cells.Cell
        - Cells.Columns
        - Cells.ColumnWidth
        - Cells.CopyFromRecordset(RecordSet, Long, Long, Long, Long, Long, String, Long)
        - Cells.GetCellAt(Long, Long)
        - Cells.GetDataFromExcelSheet(String, String, Long, Long, Long, Long, Long)
        - Cells.GetDataFromRecordset(RecordSet, Long, Long, Long, Long, Long, String, SAXLFieldListMode)
        - Cells.GroupColumns(Long, Long, Boolean)
        - Cells.GroupRows(Long, Long, Boolean)
        - Cells.HPageBreak(SACell)
        - Cells.Item
        - Cells.MergeCells(Long, Long, Long, Long)
        - Cells.PreserveStrings
        - Cells.Range
        - Cells.RowHeight

    - Cells.Rows
    - Cells.StandardHeight
    - Cells.StandardWidth
    - Cells.SummaryColumn
    - Cells.SummaryRow
    - Cells.UsedRange
    - Cells.VPageBreak(SACell)
- Characters
    - Characters.Font
    - Characters.Index
    - Characters.Length
- Chart
    - Chart.AutoScale
    - Chart.BopPopGapSize
    - Chart.BopPopSecondPieSize
    - Chart.BopPopSplitType
    - Chart.BopPopSplitValue
    - Chart.CategoryAxis
    - Chart.ChartArea
    - Chart.DataTable
    - Chart.DataTableHasHorizontalBorders
    - Chart.DataTableHasVerticalBorders
    - Chart.DataTableShowsSeriesKeys
    - Chart.FillSurfaceArea
    - Chart.Legend
    - Chart.LegendLocation
    - Chart.LegendVertical
    - Chart.Move(Long, Long, Long, Long)
    - Chart.PlotArea
    - Chart.RectangularCorner
    - Chart.SecondaryCategoryAxis
    - Chart.SecondaryValueAxis
    - Chart.SeriesCollection
    - Chart.SetChartType(ASXChartType, Integer)
    - Chart.ShowDataTable
    - Chart.ShowHorizontal
    - Chart.ShowLegend
    - Chart.ShowValueAsPercent
    - Chart.SizeOfCenterHole
    - Chart.SliceDistanceFromCenter
    - Chart.SpaceBetweenBars
    - Chart.SpaceBetweenCategories
    - Chart.StackDisplayedValues
    - Chart.Subtype
    - Chart.Title
    - Chart.Type
    - Chart.Valueaxis
    - Chart.View3D
- ChartFrame
    - ChartFrame.Area
    - ChartFrame.Border
    - ChartFrame.DataLabelPlacement
    - ChartFrame.HasShadow
    - ChartFrame.Height
    - ChartFrame.Number
    - ChartFrame.Text
    - ChartFrame.TextFont
    - ChartFrame.TextHorizontalAlignment
    - ChartFrame.TextRotationAngle
    - ChartFrame.TextVerticalAlignment
    - ChartFrame.Width
    - ChartFrame.X
    - ChartFrame.Y
- Charts
    - Charts.Add(ASXChartType, Integer, Long, Long, Long, Long)
    - Charts.Count
    - Charts.Item
- DataLabels
    - DataLabels.Count
    - DataLabels.Item
- DocumentProperties
    - DocumentProperties.Application
    - DocumentProperties.Author
    - DocumentProperties.Category
    - DocumentProperties.Clear()

    - DocumentProperties.Comments
    - DocumentProperties.Company
    - DocumentProperties.CreationDate
    - DocumentProperties.Keywords
    - DocumentProperties.LastSavedBy
    - DocumentProperties.LastSavedDate
    - DocumentProperties.LinksUpToDate
    - DocumentProperties.Manager
    - DocumentProperties.RevisionNumber
    - DocumentProperties.ScaleCrop
    - DocumentProperties.Subject
    - DocumentProperties.Title
- ExcelApplication
    - ExcelApplication.AnsiToUnicode(String, Long)
    - ExcelApplication.ContentType
    - ExcelApplication.CreateFont()
    - ExcelApplication.CreateStyle()
    - ExcelApplication.DecryptPassword
    - ExcelApplication.DocumentProperties
    - ExcelApplication.EncryptPassword
    - ExcelApplication.NamedRange(String, Long)
    - ExcelApplication.Open(Variant, Boolean)
    - ExcelApplication.Save(String, SASaveMethod, SAFileFormat, Response)
    - ExcelApplication.ShowHScrollBar
    - ExcelApplication.ShowVScrollBar
    - ExcelApplication.SpreadsheetsCreatedToday
    - ExcelApplication.Style
    - ExcelApplication.UnicodeToAnsi(String, Long)
    - ExcelApplication.Version
    - ExcelApplication.VersionEx
    - ExcelApplication.Worksheets
- Font
    - Font.Bold
    - Font.Charset
    - Font.Color
    - Font.Italic
    - Font.Name
    - Font.QueryFontDisp()
    - Font.Size
    - Font.Strikethrough
    - Font.Subscript
    - Font.Superscript
    - Font.Underline
- Line
    - Line.LineColor
    - Line.LineFormatting
    - Line.LineStyle
    - Line.LineWeight
- PageSetup
    - PageSetup.BlackAndWhite
    - PageSetup.BottomMargin
    - PageSetup.CenterFooter
    - PageSetup.CenterHeader
    - PageSetup.CenterHorizontally
    - PageSetup.CenterVertically
    - PageSetup.Draft
    - PageSetup.FirstPageNumber
    - PageSetup.FitToPagesTall
    - PageSetup.FitToPagesWide
    - PageSetup.FooterMargin
    - PageSetup.GetPrintTitles()
    - PageSetup.HeaderMargin
    - PageSetup.LeftFooter
    - PageSetup.LeftHeader
    - PageSetup.LeftMargin
    - PageSetup.Order
    - PageSetup.Orientation
    - PageSetup.PaperSize
    - PageSetup.PrintArea
    - PageSetup.PrintComments
    - PageSetup.PrintGridlines
    - PageSetup.PrintHeadings
    - PageSetup.PrintNotes
    - PageSetup.RightFooter
    - PageSetup.RightHeader

    - PageSetup.RightMargin
    - PageSetup.SetPrintTitleColumns(Long, Long)
    - PageSetup.SetPrintTitleRows(Long, Long)
    - PageSetup.TopMargin
    - PageSetup.UseZoom
    - PageSetup.Zoom
- Picture
    - Picture.AreaPattern
    - Picture.BorderLineType
    - Picture.BorderWeight
    - Picture.HasShadow
    - Picture.LowerDeltaX
    - Picture.LowerDeltaY
    - Picture.LowerRightColumn
    - Picture.LowerRightRow
    - Picture.Move(Long, Long, Long, Long)
    - Picture.UpperDeltaX
    - Picture.UpperDeltaY
    - Picture.UpperLeftColumn
    - Picture.UpperLeftRow
- Pictures
    - Pictures.Count
    - Pictures.CreateFromFile2(Long, Long, String)
    - Pictures.CreateFromFile(Long, Long, Long, Long, String)
    - Pictures.Item
- PivotField
    - PivotField.AddSubtotal(SAXlSubtotalType)
    - PivotField.AutoShowCount
    - PivotField.AutoShowField
    - PivotField.AutoShowRange
    - PivotField.AutoShowType
    - PivotField.AutoSortField
    - PivotField.AutoSortType
    - PivotField.ClearSubtotals()
    - PivotField.DataDisplayFormat
    - PivotField.DisplayName
    - PivotField.InOutlineForm
    - PivotField.IsCalculatedDataField
    - PivotField.Name
    - PivotField.NumberFormat
    - PivotField.RemoveSubtotal(SAXlSubtotalType)
    - PivotField.SummarizeBy
    - PivotField.Type
- PivotFields
    - PivotFields.Count
    - PivotFields.Item
- PivotTable
    - PivotTable.AddDataField(IPivotField, SAXlSummarizationType)
    - PivotTable.AutoFormatType
    - PivotTable.DataCollectionInRow
    - PivotTable.EmptyCellsValue
    - PivotTable.EnableDrillDown
    - PivotTable.EnableFieldDialog
    - PivotTable.EnableWizard
    - PivotTable.Fields
    - PivotTable.FieldsPerPage
    - PivotTable.FriendlyErrorValue
    - PivotTable.GrandTotalsForColumns
    - PivotTable.GrandTotalsForRows
    - PivotTable.InOutlineForm
    - PivotTable.MergeLabels
    - PivotTable.Move(Long, Long)
    - PivotTable.Name
    - PivotTable.PageLayout
    - PivotTable.PreserveFormatting
    - PivotTable.RepeatItemLabels
    - PivotTable.SetPrintTitles
    - PivotTable.ShowFriendlyErrorValue
    - PivotTable.ShowValueForEmptyCells
    - PivotTable.SubtotalHiddenPageItems
    - PivotTable.UpperLeftColumn
    - PivotTable.UpperLeftRow
- PivotTables
    - PivotTables.Count
    - PivotTables.CreatePivotTable(SARange, Long, Long)

    - PivotTables.Item
- Range
    - Range.AddHyperlink(String, String)
    - Range.Area
    - Range.AreaCount
    - Range.AutoFitHeight()
    - Range.AutoFitWidth()
    - Range.BorderAround(SABorderLineStyle, Long)
    - Range.Cell
    - Range.ColumnCount
    - Range.ColumnWidth
    - Range.FirstColumn
    - Range.FirstRow
    - Range.GetCellAt(Integer, Integer)
    - Range.GroupColumns(Long, Long, Boolean)
    - Range.GroupRows(Long, Long, Boolean)
    - Range.Intersect(SARange)
    - Range.Item
    - Range.MergeCells()
    - Range.Name
    - Range.RowCount
    - Range.RowHeight
    - Range.Style
    - Range.Union(SARange)
    - Range.UsedRange
    - Range.Value
- Series
    - Series.Area
    - Series.AxisGroup
    - Series.Border
    - Series.ChartSubType
    - Series.ChartType
    - Series.DataLabels
    - Series.ErrorBar
    - Series.ErrorBarLine
    - Series.HasSmoothedLine
    - Series.LabelFrame
    - Series.LabelShowActualValue
    - Series.LabelShowCategoryAndValueAsPercent
    - Series.LabelShowCategoryLabel
    - Series.LabelShowValueAsPercent
    - Series.MarkerFormat
    - Series.MarkerSize
    - Series.MarkerType
    - Series.Name
    - Series.ScatterValues
    - Series.ShowInLegend
    - Series.SpaceBetweenBars
    - Series.SpaceBetweenCategories
    - Series.Values
    - Series.XErrorBarCustomMinusValues
    - Series.XErrorBarCustomPlusValues
    - Series.XErrorBarTShaped
    - Series.XErrorBarValue
    - Series.XErrorBarValueSource
    - Series.YErrorBarCustomMinusValues
    - Series.YErrorBarCustomPlusValues
    - Series.YErrorBarTShaped
    - Series.YErrorBarValue
    - Series.YErrorBarValueSource
- SeriesCollection
    - SeriesCollection.Add(Area, , Long)
    - SeriesCollection.CategoryData
    - SeriesCollection.Item
    - SeriesCollection.LabelShowActualValue
    - SeriesCollection.LabelShowCategoryAndValueAsPercent
    - SeriesCollection.LabelShowCategoryLabel
    - SeriesCollection.LabelShowValueAsPercent
- Style
    - Style.BackgroundColor
    - Style.BorderColor
    - Style.BorderLineStyle
    - Style.Font
    - Style.ForegroundColor
    - Style.Hidden

        - Style.HorizontalAlignment
        - Style.Indent
        - Style.Locked
        - Style.Name
        - Style.Number
        - Style.Orientation
        - Style.Pattern
        - Style.ShrinkToFit
        - Style.VerticalAlignment
        - Style.WrapText
    - Worksheet
        - Worksheet.Cells
        - Worksheet.Charts
        - Worksheet.Copy()
        - Worksheet.FreezePanes(SACell)
        - Worksheet.Name
        - Worksheet.PageSetup
        - Worksheet.Pictures
        - Worksheet.PivotTables
        - Worksheet.SelectSheet()
        - Worksheet.ShowGridlines
        - Worksheet.ShowRowColHeaders
        - Worksheet.Visible
        - Worksheet.WriteProtect
        - Worksheet.Zoom
    - Worksheets
        - Worksheets.Add(String)
        - Worksheets.Count
        - Worksheets.Item
        - Worksheets.Remove(Variant)
- Formula Functions
    - Database Functions
    - Financial Functions
    - Information Functions
    - Logical Functions
    - Lookup and Reference Functions
    - Math Functions
    - Statistical Functions
    - String Functions
    - Time Functions
- Formula Calculation Operators
- Formatting Codes
- Chart Codes
- PivotTable AutoFormat Types

Object Model

ExcelWriter includes three separate components:

- ExcelTemplate

  Introduction to ExcelTemplate: Create a Spreadsheet with ExcelTemplate

  The ExcelTemplate represents a template spreadsheet. This object opens an ExcelWriter template file, populates it with data from a specified data source, and generates a new Excel spreadsheet.

- ExcelApplication

  ExcelApplication is not available in ExcelWriterSE.

  Introduction to ExcelApplication: Create a Spreadsheet with ExcelApplication

  The ExcelApplication object represents an Excel spreadsheet. This object creates several additional objects, each of which represents some aspect of an Excel spreadsheet. ExcelApplication's sub-objects are:

    - Area
    - Axis
    - Cell
    - Cells
    - Chart

- ChartFrame
- Charts
- DocumentProperties
- Font
- Line
- PageSetup
- Picture
- Pictures
- PivotField
- PivotFields
- PivotTable
- PivotTables
- Range
- Series
- SeriesCollection
- Style
- 3DProperties
- Worksheet
- Worksheets

ExcelTemplate Programmer's Reference

The ExcelTemplate object represents a **template** workbook - 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.

To create an instance of the ExcelTemplate object, use:

```
Set objTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")
```

> The ExcelTemplate object supports Excel's BIFF8 (Excel 97/2000/XP) format only; ExcelTemplate does not support BIFF7 (Excel 95) format templates and will not generate BIFF7 format files. The ExcelApplication object can produce either Excel 97/2000/XP (BIFF8) format files or Excel 95 (BIFF7) format files.

- ExcelTemplate — The ExcelTemplate object represents a template workbok - 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.

ExcelTemplate

Description

The ExcelTemplate object represents a template workbok - 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.

Signature: VB.NET

```
Public Class ExcelTemplate
```

Remarks

> The ExcelTemplate object supports Excel's BIFF8 (Excel 97/2000/XP) format only; ExcelTemplate does not support BIFF7 (Excel 95) format templates, and will not generate BIFF7 format files. (The ExcelApplication object can produce either Excel 97/2000/XP (BIFF8) format files or Excel 95 (BIFF7) format files.)

Examples

vbscript

```
Set objTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")
```

Properties

Name	Description
ContentType	Sets or returns the generated spreadsheet's MIME content type.
DataSource	Sets a data source to bind to template data markers.
DecryptPassword	Set DecryptPassword to decrypt an encrypted template.
EncryptPassword	If you set EncryptPassword, ExcelWriter will encrypt the generated spreadsheet using RC4 encryption.
PreserveStrings	Sets whether ExcelWriter will insert all data source values into the template as strings or try to convert non-string values (numbers, dates, etc.) to Excel data types.
SpreadsheetsCreatedToday	Returns the number of spreadsheets ExcelWriter created so far in the current 24 hour period.

Methods

Name	Description
AnsiToUnicode(String, Long)	Converts an Ansi string to a Unicode string.
Open(String)	Opens an ExcelWriter template spreadsheet.
Process(String, SAProcessMethod, Long)	Binds data to the template and generates a new spreadsheet
UnicodeToAnsi(String, Long)	Converts a Unicode string to an Ansi string.

ExcelTemplate.AnsiToUnicode(String, Long)

Description

Converts an Ansi string to a Unicode string.

Signature: VB.NET

```
Public Function AnsiToUnicode(Ansi As String, CodePage As Long) Unicode As String
```

Parameters

Ansi

The ANSI String to convert

CodePage

The integer representing the code page of the current language.

Remarks

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.

The AnsiToUnicode method takes a non-Latin alphabet string and its language's code page and returns a Unicode string. If the client HTML script's charset property is set to the string's language or to UTF-8, and the string is converted to Unicode using AnsiToUnicode, the string will be displayed correctly in the generated spreadsheet.

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

Examples

vbscript

```
'--- The following lines get a Hebrew string from an HTML form, convert the
'--- string to Unicode, and insert the string at the FirstName data marker.
'--- The example uses a specific language code page; to correctly display
'--- strings in any language, use code page 65001.
Set xlw = Server.CreateObject("Softartisans.ExcelTemplate")
HebrewString = xlw.ansitounicode(Request.Form("FirstName"), 1255)
xlw.Datasource("FirstName") = HebrewString
```

ExcelTemplate.ContentType

Description

Sets or returns the generated spreadsheet's MIME content type.

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

vbscript

```
Set objSAXLTmplt = Server.CreateObject("SoftArtisans.ExcelTemplate")
...
Set Recordset = adoConnect.Execute(sqlText)
objSAXLTmplt.Open Server.MapPath(Application("vroot") &_
   "templates/simpletemplate.xls")
objSAXLTmplt.DataSource("Recordset") = Recordset
objSAXLTmplt.ContentType = "application/unknown"
objSAXLTmplt.Process "TemplateSample.xls", saProcessOpenInPlace
```

ExcelTemplate.DataSource

Description

Sets a data source to bind to template data markers.

Signature: VB.NET

```
Public Property DataSource(Name As Variant, MaxRows As Long, DataSource As Variant)
```

Remarks

The data source may be an ADO Recordset, a variable, a one dimensional array or a two dimensional array. A data marker is a string beginning with **%%=** (or **%%=$** for variables and arrays), which is placed in a cell of the template spreadsheet. Each data marker specifies a database field, variable, or array column to insert in the spreadsheet column containing the marker. Every data marker in the template spreadsheet must refer to a DataSource which has been assigned in the ExcelWriter script.

You may assign many DataSources to a single spreadsheet.

DataSources should be set after all data manipulation is complete and before calling the Process method.

DataSource takes two optional parameters:

- **Name**
  The data source name specified in the data marker (e.g. "Orders" in **%%=Orders.OrderID**).
  The string passed to the Name parameter must begin with a letter.
  Default value: "" (empty string). If you wish to use binding by order,
  you must leave the name parameter empty.

- **MaxRows**
  The maximum number of rows to return from a database or 2-dimensional array.
  Default value: 65536.

Data marker formats	
Recordset data marker	**%%=[DataSourceNameOrNumber].FieldNameOrNumber**
Variable data marker	**%%=$DataSourceNameOrNumber**
One-Dimensional Array data marker	**%%=$DataSourceNameOrNumber**
Two-dimensional Array data marker	**%%=$DataSourceNameOrNumber.#ColumnNumber**

Important:

- In ExcelWriter version 3.1 and higher, when binding a DataSource by position, if the number of fields in a particular recordset is not known in advance, you may include excess datamarkers in the template, and ExcelWriter will remove any that are not used.

- DataSource order numbers, and field position numbers, are 1-based. If ExcelWriter encounters **%%=#0.[field]** or **%%=[DataSource].#0** in a data marker, an error will occur.

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

DataSource Formats for Binding to Recordset			
	Format	**Data Marker**	**Code**
1	**Generic**	**%%=DataSourceName.FieldName**	obj.Datasource("DataSourceName") = MyDataSource
2	**Implicit bind**	**%%=FieldName** If there are many datasources assigned, implicit bind assumes the first.	obj.Datasource = MyDataSource
3	**Bind by order**	**%%=#1.FieldName** **%%=#2.FieldName**	obj.Datasource = MyDataSource obj.Datasource = MyDataSource1
4	**Bind by field position**	**%%=DataSourceName.#1** **%%=DataSourceName.#2**	obj.Datasource("DataSourceName") = MyDataSource

2 & 3	**Implicit Bind with Bind by order**	**%%=FieldName** **%%=#2.FieldName**	obj.Datasource = MyDataSource obj.Datasource = MyDataSource1
1,2,3 & 4	**Generic, Implicit, Bind by order, and Bind by field position**	**%%=#1** **%%=#2.#2**	obj.Datasource = MyDataSource obj.Datasource = MyDataSource1

In ExcelWriter version 3.1 and higher, DataSource sequential numbers are 1-based; when binding by order the first data marker is **%%=#1[.field]**. If ExcelWriter encounters a **%%=#0[.field]** data marker, an error will occur.

In ExcelWriter version 3.1 and higher, Field position sequential numbers are 1-based. If ExcelWriter encounters a data marker containing field 0 (e.g. **%%=DataSource.#0**), an error will occur.

In ExcelWriter version 3.1 and higher, when binding a DataSource by position, if the number of fields in a particular recordset is not known in advance, you may include excess datamarkers in the template, and ExcelWriter will remove any that are not used.

DataSource formats for binding to Variables

	Format	Data Marker	Code
1	**Generic**	**%%=$DataSourceName**	obj.Datasource("DataSourceName") = MyVar
2	**Bind by order**	**%%=$#1** **%%=$#2**	obj.Datasource = MyVar obj.Datasource = MyVar1
1 & 2	**Generic Bind with Bind by order**	**%%=$DataSourceName** **%%=$#2**	obj.Datasource("DataSourceName") = MyVar1 obj.Datasource = MyVar2

As part of the bind to variable functionality, it is possible to directly assign a string to a datasource. ExcelWriter will automatically convert it to a variant. For example:
obj.Datasource("DataSourceName") = "hello world"

DataSource formats for binding to One-Dimensional Arrays

	Format	Data Marker	Code
1	**Generic**	%%=$DataSourceName	obj.Datasource("DataSourceName") = MyArray
2	**Bind by order**	**%%=$#1** **%%=$#2**	obj.Datasource = MyArray1 obj.Datasource = MyArray2
1 & 2	**Generic Bind with Bind by order**	**%%=$DataSourceName** **%%=$#2**	obj.Datasource("DataSourceName") = MyArray1 obj.Datasource = MyArray2

The data from a one-dimensional array gets loaded into rows beginning at the cell with the corresponding data marker, i.e. for each element of the array, a new row is inserted. If you wish to accomplish the equivalent of displaying a one-dimensional array horizontally across columns, you should create a two-dimensional array with only one row and place datamarkers horizontally.

DataSource formats for binding to Two-Dimensional Arrays

	Format	Data Marker	Code
1	**Generic**	**%%=$DataSourceName.#1**	Dim TwoDArray(5,10) obj.Datasource("DataSourceName") = TwoDArray

2	**Bind by order**	**%%=$#1.#1** **%%=$#2.#1**	Dim TwoDArray1(5, 10) obj.Datasource = TwoDArray1 Dim TwoDArray2(12, 3) obj.Datasource = TwoDArray2
1 & 2	**Generic Bind with Bind by order**	**%%=$DataSourceName.#1** **%%=$#2.#1**	Dim TwoDArray1(5, 10) obj.Datasource("DataSourceName") = TwoDArray1 Dim TwoDArray2(12, 3) obj.Datasource = TwoDArray2

In ExcelWriter version 3.1 and higher, DataSource order and column position numbers are 1-based. If ExcelWriter encounters data markers with **%%=#0.#columnNumber** or **%%=DataSource.#0**, an error will occur. Column position numbers in the data markers must begin with #1, even if the lower bound of the array is other than 1, as in: Dim TwoDArray(1 to 5, 40 to 45)

ExcelWriter imports data in the order row, column. Therefore, two-dimensional arrays must be created with rows for the first dimension and columns for the second dimension. For example: Dim myArray(5,10) will fill 6 rows, 11 columns (VBScript arrays are 0-based by default).

Array columns must always be bound by position, because unlike recordsets, they do not have field names:
%%=$MyArray.#1
%%$#3.#2

In ExcelWriter version 3.1 and higher, if the number of columns in a particular 2-D array is not known in advance, one may put as many datamarkers as desired in the template spreadsheet, and ExcelWriter will remove any that are not used.

Examples

vbscript

```
'--- Setting Name and MaxRows.
'--- Name must begin with a letter.
'--- This format cannot be used with binding by order
obj.DataSource("Name", 50) = MyDataSource

'--- Setting Name only.
'--- Name must begin with a letter
'--- This format cannot be used with binding by order
obj.DataSource("Name") = MyDataSource

'--- Setting MaxRows only. Use any of the following formats.
obj.DataSource (, 50)
obj.DataSource (50)

'--- If neither parameter is set, default settings will be used.
obj.DataSource = MyDataSource
```

When the name parameter is not set, the DataSource should be referenced by number in your data markers, according to the order of DataSource assignments. See binding by order.

DataSources with different formats and different types of data may be used together the same script.
For example:

vbscript

```
obj.DataSource = myVariable
obj.DataSource("Orders", 200) = myRecordset
obj.Datasource("twoD") = myArray
obj.Datasource = anotherRecordset
```

ExcelWriter expects data in the order row, column. Therefore all 2-dimensional arrays must be created to take rows for the first dimension and columns for the second dimension. For example, if you want to create an array in VBScript with 4 rows and 2 columns, you should declare it as follows:

vbscript

```
dim myArray(3, 1)

'--- VBScript arrays are 0-based
'--- Alternatively you can set the lower and upper bound:
'dim myArray(1 to 4, 1 to 2)
```

ExcelTemplate.DecryptPassword

Description

Set DecryptPassword to decrypt an encrypted template.

Signature: VB.NET

```
Public Property DecryptPassword As String
```

Remarks

If you set EncryptPassword, ExcelWriter will encrypt the generated spreadsheet - when Process is called - using RC4 encryption. 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

vbscript

```
Set XlwTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")
XlwTemplate.EncryptPassword = "MyPassword"
```

ExcelTemplate.EncryptPassword

Description

If you set EncryptPassword, ExcelWriter will encrypt the generated spreadsheet using RC4 encryption.

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

vbscript

```
Set XlwTemplate = Server.CreateObject("SoftArtisans.ExcelTemplate")
XlwTemplate.EncryptPassword = "MyPassword"
```

ExcelTemplate.Open(String)

Description

Opens an ExcelWriter template spreadsheet.

Signature: VB.NET

```
Public Void Open(FileName As String)
```

Parameters

FileName

The file path and name of the template file to open, as a String

Remarks

Opens an ExcelWriter template spreadsheet - a file created in Microsoft Excel that contains data markers where values will be inserted. A data marker is a cell value beginning with "%%=" that specifies a database column, variable, or array to insert in the spreadsheet column containing the marker.

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

> The ExcelTemplate object supports Excel's BIFF8 (Excel 97/2000/XP) format only; ExcelTemplate does not support BIFF7 (Excel 95) format templates, and will not generate BIFF7 format files.

Examples

vbscript

```
objTemplate.Open "c:\templates\template.xls"
```

ExcelTemplate.PreserveStrings

Description

Sets whether ExcelWriter will insert all data source values into the template as strings or try to convert non-string values (numbers, dates, etc.) to Excel data types.

Signature: VB.NET

```
Public Property PreserveStrings As Boolean
```

Remarks

To preserve a numeric string as a string, set PreserveStrings to true. PreserveStrings is set to False by default.

PreserveStrings is a Read/Write property.

> When PreserveStrings is not enabled, data source values like Social Security numbers and zip codes will be converted to numbers and leading zeros will be lost.

Examples

vbscript

```
objTemplate.PreserveStrings = True
```

ExcelTemplate.Process(String, SAProcessMethod, Long)

Description

Binds data to the template and generates a new spreadsheet

Signature: VB.NET

```
Public Function Process(FileName As String, ProcessMethod As SAProcessMethod, ExcludeMacro As Long)
Application As ExcelApplication
```

Parameters

FileName

Path and file name of the new spreadsheet as a String

ProcessMethod

Optional. Default value is 0 (saProcessDefault). SAProcessMethod object that specifies whether to save the file to disk, open it in Excel, open it in the browser, or return an ExcelApplication object.

ExcludeMacro

Optional. Default value is False. If set to True, macros in the template will be excluded from the generated spreadsheet.

When ExcludeMacro is enabled, buttons associated with macros may produce the error "Data may have been lost." Therefore, when excluding macros, remove all associated buttons from the spreadsheet.

Remarks

Enters data source values in the template and generates a new spreadsheet.

Assign the ProcessMethod parameter by name or number:

0	**saProcessDefault**	Save the file to disk
1	**saProcessOpenInExcel**	Open the file in Excel
2	**saProcessOpenInPlace**	Open in the browser
3	**saProcessOpenForScripting**	Return the file in memory as an ExcelApplication object. This value is not available in ExcelWriterSE.

Examples

vbscript

```
objTemplate.Process "file.xls", saProcessOpenInPlace
```

ExcelTemplate.SpreadsheetsCreatedToday

Description

Returns the number of spreadsheets ExcelWriter created so far in the current 24 hour period.

Signature: VB.NET

```
Public ReadOnly Property SpreadsheetsCreatedToday As Integer
```

Remarks

Note that SpreadsheetsCreatedToday contains the number of spreadsheets created by both ExcelApplication and ExcelTemplate.

The hour count begins when the first spreadsheet is created. The count is reset when the first spreadsheet is created after the previous 24 hour period. For example, if you create your first spreadsheet at 10:00 AM on June 2, the 24 hour period will end at 10:00 AM on June 3, and the next period will begin when you create the first spreadsheet after 10:00 AM on June 3.

ExcelTemplate.UnicodeToAnsi(String, Long)

Description

Converts a Unicode string to an Ansi string.

Signature: VB.NET

```
Public Function UnicodeToAnsi(Unicode As String, CodePage As Long) Ansi As String
```

Parameters

Unicode

Unicode value you wish to convert to ANSI, as a string

CodePage

Integer value corresponding to the codepage you wish to use in the conversion

Remarks

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.

To correctly display strings in only one language, uses a specific language code page. To correctly display strings in any language, use code page 65001. For a complete list of charset and code page values see, Character Set Recognition.

Examples

vbscript

```
'--- UnicodeToAnsi takes a Unicode string and the string language's code page and returns an Ansi
string
AnsiString = UnicodeToAnsi(UnicodeString, CodePage)
```

ExcelApplication Programmer's Reference

ExcelApplication is ExcelWriter's principal object. The ExcelApplication object represents an Excel workbook. To generate an Excel spreadsheet, you must create an instance of the ExcelApplication object.

To create an instance of the ExcelApplication object in an ASP script use,VBScript
Set objExcelApplication = Server.CreateObject("Softartisans.ExcelWriter")

- 3DProperties — The 3DProperties object represents the set of properties specific to three-dimensional charts.
- Area — The Area object represents the appearance of a chart area.
- Axis — Represents a chart's X, Y, or Z axis.
- Cell — Represents a single cell in a worksheet.
- Cells — Represents the set of cells in a worksheet.
- Characters — Represents a range of characters within a cell.
- Chart — Represents a single chart in a worksheet.
- ChartFrame — ChartArea, DataTable, Legend, LabelFrame, PlotArea, and Title are ChartFrame objects.
- Charts — The collection of charts in a worksheet.
- DataLabels — Represents the collection of data labels for a series.
- DocumentProperties — Represents properties of a specified spreadsheet.
- ExcelApplication — Represents an Excel Workbook in ExcelWriter.
- Font — Represents a font style that you can use when creating a reusable Style object, or a single-cell Format.
- Line — Represents the appearance of a chart line.
- PageSetup — Represents the page layout properties of a printed worksheet.
- Picture — Represents a single picture in a spreadsheet.
- Pictures — The collection of all pictures in a worksheet.
- PivotField — Represents a single field in a PivotTable.
- PivotFields — The collection of fields in a PivotTable.
- PivotTable — Represents a single PivotTable in a worksheet.
- PivotTables — The collection of PivotTables in a worksheet.
- Range — Represents a range of cells within a worksheet.
- Series — Represents a data series in a chart.
- SeriesCollection — The collection of data series in a single chart.
- Style — Represents a text style that can be assigned to one cell or a range of cells.
- Worksheet — Represents a single Excel worksheet in a workbook.
- Worksheets — The collection of worksheets in a workbook.

3DProperties

Description

The 3DProperties object represents the set of properties specific to three-dimensional charts.

Signature: VB.NET

```
Public Class 3DProperties
```

Remarks

To create a 3DProperties object, use Chart.View3D.

Examples

vbscript

```
Set 3DProperties = chart.view3d
```

Properties

Name	Description
ChartDepth	Sets or retrieves the depth of a three-dimensional chart. The default value of ChartDepth is 100.
DistanceFromEyeToChart	Sets or retrieves a perspective value, from 0 to 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 retrieves the elevation angle of a chart.
PlotVolumeHeight	Sets or retrieves the height of a 3D chart as a percentage of the chart's base length.

RotationAngle	Sets or retrieves the rotation angle of a three-dimensional chart.
SeriesGapDepth	Sets or retrieves the gap width between categories of series values.
Show3D	Sets whether to display the chart with 3-dimensional effect.
Stack3DColumns	Sets whether to display series categories from front to back or side to side.
UseAutoScaling	Sets whether to automatically scale the chart size or to use PlotVolumeHeight to determine the size of the chart.
UsePerspectiveTransform	Sets whether to use the value of DistanceFromEyeToChart to control the appearance of chart depth.
ZAxis	Returns an Axis object representing a three-dimensional chart's z-axis.

3DProperties.ChartDepth

Description

Sets or retrieves the depth of a three-dimensional chart. The default value of ChartDepth is 100.

Signature: VB.NET

```
Property ChartDepth As Integer (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Set the chart depth to 200
MyChart.View3D.ChartDepth = 200
```

3DProperties.DistanceFromEyeToChart

Description

Sets or retrieves a perspective value, from 0 to 100 degrees, that represents the ratio of the chart's front to its back. The perspective value determines the appearance of chart depth.

Signature: VB.NET

```
Property DistanceFromEyeToChart As Integer (read/write)
```

Remarks

The default value of DistanceFromEyeToChart is 30.

Setting DistanceFromEyeToChart will have no effect unless you enable UsePerspectiveTransform.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Enable UsePerspectiveTransform
MyChart.View3D.UsePerspectiveTransform = True

'--- Set the distance to 80
MyChart.View3D.DistanceFromEyeToChart = 80
```

3DProperties.ElevationAngle

Description

Sets or retrieves the elevation angle of a chart.

Signature: VB.NET

```
Property ElevationAngle As Integer (read/write)
```

Remarks

By default, ElevationAngle is set to 15 – that is, the chart is elevated 15 degrees.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Set elevation angle to 45 degrees
MyChart.View3D.ElevationAngle = 45
```

3DProperties.PlotVolumeHeight

Description

Sets or retrieves the height of a 3D chart as a percentage of the chart's base length.

Signature: VB.NET

```
Property PlotVolumeHeight As Integer (read/write)
```

Remarks

By default, PlotVolumeHeight is 100 – that is, 100% of the base length.

Setting PlotVolumeHeight will have no effect unless you disable UseAutoScaling.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Disable UseAutoScaling
MyChart.View3D.UseAutoScaling = False

'--- Set the chart height to 50% of the base length.
MyChart.View3D.PlotVolumeHeight = 50
```

3DProperties.RotationAngle

Description

Sets or retrieves the rotation angle of a three-dimensional chart.

Signature: VB.NET

```
Property RotationAngle As Integer (read/write)
```

Remarks

By default, RotationAngle is set to 20 – that is, the chart is rotated 20 degrees.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Set the rotation angle of a chart to 60 degrees.
MyChart.View3D.RotationAngle = 60
```

3DProperties.SeriesGapDepth

Description

Sets or retrieves the gap width between categories of series values.

Signature: VB.NET

```
Property SeriesGapDepth As Integer(read/write)
```

Remarks

SeriesGapDepth values range from 0 to 500. The greater the value of SeriesGapDepth, the wider the gap. SeriesGapDepth is set to 150 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Set the series gap depth to 200
MyChart.View3D.SeriesGapDepth = 200
```

3DProperties.Show3D

Description

Sets whether to display the chart with 3-dimensional effect.

Signature: VB.NET

```
Property Show3D As Boolean (read/write)
```

Remarks

When set to True, displays chart with three-dimensional effect. When set to False, displays two dimensions, even if the chart has a 3-D subtype. Show3D is set to True by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Display the 3D chart in 2D
MyChart.View3D.Show3D = False
```

3DProperties.Stack3DColumns

Description

Sets whether to display series categories from front to back or side to side.

Signature: VB.NET

```
Property Stack3DColumns As Boolean (read/write)
```

Remarks

When set to False, displays series categories from front to back of the chart, with a gap between the columns. When set to True, displays columns side by side.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Display columns side-by-side
MyChart.View3D.Stack3DColumns = True
```

3DProperties.UseAutoScaling

Description

Sets whether to automatically scale the chart size or to use PlotVolumeHeight to determine the size of the chart.

Signature: VB.NET

```
Property UseAutoScaling As Boolean (read/write)
```

Remarks

When UseAutoScaling is enabled, Microsoft Excel will automatically scale a three-dimensional chart to be close to the size of its two-dimensional version. UseAutoScaling is set to 1, that is, enabled, by default. Disable UseAutoScaling to use PlotVolumeHeight.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Disable auto scaling
MyChart.View3D.UseAutoScaling = False
```

3DProperties.UsePerspectiveTransform

Description

Sets whether to use the value of DistanceFromEyeToChart to control the appearance of chart depth.

Signature: VB.NET

```
Property UsePerspectiveTransform As Boolean (read/write)
```

Remarks

When enabled, allows you to control the appearance of chart depth, by setting the value of DistanceFromEyeToChart. UsePerspectiveTransform is disabled by default. When UsePerspectiveTransform is disabled, the chart's axes will be displayed at right angles, regardless of rotation and elevation values.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Enable UsePerspectiveTransform
MyChart.View3D.UsePerspectiveTransform = True

'--- Set the distance to 80
MyChart.View3D.DistanceFromEyeToChart = 80
```

3DProperties.ZAxis

Description

Returns an Axis object representing a three-dimensional chart's z-axis.

Signature: VB.NET

```
Property ZAxis As IAxis (read-only)
```

Remarks

The properties of ZAxis are AxisLine, LabelFont, LabelPosition, MajorGridline, MajorIncrement, MajorTickType, MinorGridline, MinorIncrement, MinorTickType, Number, Title, and ValuesInReverseOrder.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a 3D column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   6, 1, 1, 10, 10)

'--- Use ZAxis to get an IAxis interface to adjust properties of the Z axis
'--- Set the major tick type of the axis
Set AxisZ = MyChart.View3D.ZAxis
AxisZ.MajorTickType = saxlsOutsideLineTick
```

Area

Description

The Area object represents the appearance of a chart area.

Signature: VB.NET

```
Public Class Area
```

Remarks

ExcelWriter includes the following area objects:

- Series Area
- MarkerFormat
- Wall3D
- ChartFrame Area

Properties

Name	Description
AreaFormatting	Sets or retrieves the type of formatting applied to an Area object.
BackgroundColor	Sets or retrieves the background color of an Area object (Series Area, MarkerFormat, Wall3D, or ChartFrame Area).
ForegroundColor	Sets or retrieves the foreground color of an Area object (Series Area, MarkerFormat, Wall3D, or ChartFrame Area).
Pattern	Sets or retrieves the background pattern of an Area object (Series Area, MarkerFormat, Wall3D, or ChartFrame Area).

Area.AreaFormatting

Description

Sets or retrieves the type of formatting applied to an Area object.

Signature: VB.NET

```
Property AreaFormatting As ASXCreationType (read/write)
```

Remarks

The three types of formatting and their codes are

saxlsCreationAutomatic	**0**
saxlsCreationNone	**1**
saxlsCreationCustom	**2**

When AreaFormatting is set to saxlsCreationAutomatic (0), the default formatting settings will be applied to the Area object. SaxlsCreationCustom (2) enables custom formatting assigned to the Area object (colors or pattern). SaxlsCreationNone will remove all Area formatting (default or custom).

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart,0,2,2,10,20)

'--- The IChartFrame.Area property returns an SAArea object
'--- Remove chart area formatting by setting saxlsCreationNone
MyChart.CharArea.Area.AreaFormatting = saxlsCreationNone
```

Area.BackgroundColor

Description

Sets or retrieves the background color of an Area object (Series Area, MarkerFormat, Wall3D, or ChartFrame Area).

Signature: VB.NET

```
Property BackgroundColor As Integer (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and create
'--- a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 2, 2, 10, 20)

'--- Use an SAArea object to set the background
'--- color of a chart
MyChart.CharArea.Area.BackgroundColor = RGB(255, 123, 100)
```

Area.ForegroundColor

Description

Sets or retrieves the foreground color of an Area object (Series Area, MarkerFormat, Wall3D, or ChartFrame Area).

Signature: VB.NET

```
Property ForegroundColor As Integer (read/write)
```

Remarks

Use RGB values to assign colors.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 2, 2, 10, 20)

'--- Use an SAArea object to set the foreground
'--- color of a chart
MyChart.CharArea.Area.ForegroundColor = RGB(255, 123, 100)
```

Area.Pattern

Description

Sets or retrieves the background pattern of an Area object (Series Area, MarkerFormat, Wall3D, or ChartFrame Area).

Signature: VB.NET

```
Property Pattern As SAXAreaPatternType (read/write)
```

Remarks

ExcelWriter supports nineteen patterns, listed in the table below.

No Pattern	0
0% Pattern	1
50% Pattern	2
70% Pattern	3
25% Pattern	4
Dark Horizontal	5
Dark Vertical	6
Dashed Downward Diagonal	7
Dashed Upward Diagonal	8
Small Checkerboard	9
Trellis	10
Light Horizontal	11
Light Vertical	12
Wide Downward diagonal	13
Dark Upward diagonal	14
Small Grid	15
60% Pattern	16
20% Pattern	17
10% Pattern	18

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 2, 2, 10, 20)

'--- Set the pattern for an SAArea object to 50%
MyChart.CharArea.Area.Pattern = 2
```

Axis

Description

Represents a chart's X, Y, or Z axis.

Signature: VB.NET

```
Public Class Axis
```

Remarks

ExcelWriter includes three Axis objects: CategoryAxis, ValueAxis, and, ZAxis.

Examples

vbscript

```
'--- To create a CategoryAxis object
Set CategoryAxis = chart.CategoryAxis

'--- To create a ValueAxis object
Set ValueAxis = chart.ValueAxis

'--- To create a ZAxis object
Set ZAxis = chart.View3D.ZAxis
```

Properties

Name	Description
AxisCrossesMidCategory	Determines whether category major gridlines will be between categories or in the middle of categories.
AxisLine	Returns a Line object representing the appearance of an Axis.
CategoryAxisCrossMaxValue	Sets where the category axis will cross the value axis
CategoryCrossingValue	Sets or retreives the Y value at which the category axis crosses the value axis.
LabelFont	Sets or returns a Font object representing the label font for the specified axis.
LabelPosition	Sets or returns the position of axis labels.
LogarithmicScale	Sets whether to convert the chart into a logarithmic chart
MajorGridline	Returns a Line object representing the major gridlines of an axis.
MajorIncrement	Sets or returns the major unit of measurement of an axis – that is, the gap between its major gridlines and/or major ticks marks.
MajorTickType	Determines the display of axis tick marks on the major gridlines.
MaxValue	Sets or returns the maximum value on the value axis.

MinorGridline	Returns a Line object representing the minor gridlines of an axis.
MinorIncrement	Sets or returns the minor unit of measurement of an axis – that is, the gap between its minor gridlines and/or minor ticks marks.
MinorTickType	Determines the display of axis tick marks on the minor gridlines.
MinValue	Sets or returns the minimum value on the value axis.
Number	Sets the display format of axis values.
TextRotationAngle	Sets or returns the rotation angle of axis values.
Title	Returns a ChartFrame object representing the axis title.
ValuesInReverseOrder	Determines whether to display axis values in reverse order.
Wall3D	Returns an Area object representing the walls or floor of a three-dimensional chart.
Wall3DBorder	Returns a Line object representing the borders of a 3-D chart's walls or floor.

Axis.AxisCrossesMidCategory

Description

Determines whether category major gridlines will be between categories or in the middle of categories.

Signature: VB.NET

```
Property AxisCrossesMidCategory As Boolean (read/write)
```

Remarks

Valid only for the category axis.

By default, AxisCrossesMidCategory is set to True and category major gridlines are between categories.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set AxisCrossesMidCategory to False for the category axis
MyChart.CategoryAxis.AxisCrossesMidCategory = False
```

Axis.AxisLine

Description

Returns a Line object representing the appearance of an Axis.

Signature: VB.NET

```
Property AxisLine As SALine (read-only)
```

Remarks

The properties of AxisLine are LineColor, LineFormatting, LineStyle, and LineWeight.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- The AxisLine property returns SALine
'--- Use it to format the line for the axis
'--- Set the color to red
MyChart.CategoryAxis.AxisLine.LineColor = RGB(255, 0, 0)
```

Axis.CategoryAxisCrossMaxValue

Description

Sets where the category axis will cross the value axis

Signature: VB.NET

```
Property CategoryAxisCrossMaxValue As Boolean (read/write)
```

Remarks

Property of the ValueAxis or the SecondaryValueAxis. When set to true, the category axis will cross the value axis at its maximum value.

When CategoryAxisCrossMaxValue is a property of the value axis, it is set to false by default. When CategoryAxisCrossMaxValue is a property of the secondary value axis, it is set to true by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set CategoryAxisCrossMaxValue = True
MyChart.ValueAxis.CategoryAxisCrossMaxValue = True
```

Axis.CategoryCrossingValue

Description

Sets or retreives the Y value at which the category axis crosses the value axis.

Signature: VB.NET

```
Property CategoryCrossingValue As Double (read/write)
```

Remarks

Property of the ValueAxis.

CategoryCrossingValue is 0 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Sets the Y value at which the category
'--- axis crosses the value axis to 5
MyChart.ValueAxis.CategoryCrossingValue = 5
```

Axis.LabelFont

Description

Sets or returns a Font object representing the label font for the specified axis.

Signature: VB.NET

```
Property LabelFont As SAFont (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series, MyFont
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Create a new font
Set MyFont = XlwApp.CreateFont
MyFont.Name = "Tahoma"

'--- Set the font for the label
MyChart.ValueAxis.LabelFont = MyFont
```

Axis.LabelPosition

Description

Sets or returns the position of axis labels.

Signature: VB.NET

```
Property LabelPosition As ASXLabelPosition (read/write)
```

Remarks

The values of LabelPosition are described in the following table:

Invisible Label	0
Low End Of PlotArea	1
High End Of PlotArea	2
Next To Axis	3

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series, MyFont
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the label position to "next to axis"
MyChart.ValueAxis.LabelPosition = saxlsNextToAxis
```

Axis.LogarithmicScale

Description

Sets whether to convert the chart into a logarithmic chart

Signature: VB.NET

```
Property LogarithmicScale As Integer (read/write)
```

Remarks

Property of the ValueAxis. When set to True, LogarithmicScale will:

- Convert the Y axis MinValue, MaxValue, MajorIncrement, and MinorIncrement values to powers of 10.
- Remove zeros and negative values from the chart.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series, MyFont
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Make the axis into a logarithmic scale
MyChart.ValueAxis.LogarithmicScale = True
```

Axis.MajorGridline

Description

Returns a Line object representing the major gridlines of an axis.

Signature: VB.NET

```
Property MajorGridLine As SALine (read-only)
```

Remarks

The properties of MajorGridline are LineColor, LineFormatting, LineStyle, and LineWeight.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series, MyFont
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Format the major grid line of the value axis
MyChart.ValueAxis.MajorGridLine.LineColor = RGB(255, 0, 0)
```

Axis.MajorIncrement

Description

Sets or returns the major unit of measurement of an axis – that is, the gap between its major gridlines and/or major ticks marks.

Signature: VB.NET

```
Property MajorIncrement As Double (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the value axis's major increment to 4
MyChart.ValueAxis.MajorIncrement = 4
```

Axis.MajorTickType

Description

Determines the display of axis tick marks on the major gridlines.

Signature: VB.NET

```
Property MajorTickType As ASXTickType (read/write)
```

Remarks

Tick marks are marks of measurement on an axis. The values of MajorTickType are:

Invisible Tick	0
Inside Line Tick	1
Outside Line Tick	2
Cross Axis Line Tick	3

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
  0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the major tick type of the value axis
MyChart.ValueAxis.MajorTickType = saxlsOutsideLineTick
```

Axis.MaxValue

Description

Sets or returns the maximum value on the value axis.

Signature: VB.NET

```
Property MaxValue As Double (read/write)
```

Remarks

MaxValue is a property of ValueAxis, and not of CategoryAxis or ZAxis.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
    0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the max value of the value axis
MyChart.ValueAxis.MaxValue = 12
```

Axis.MinorGridline

Description

Returns a Line object representing the minor gridlines of an axis.

Signature: VB.NET

```
Property MinorGridLine As SALine (read-only)
```

Remarks

The properties of MinorGridline are LineColor, LineFormatting, LineStyle, and LineWeight.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series, MyFont
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
  0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Format the minor grid line of the value axis
MyChart.ValueAxis.MinorGridLine.LineColor = RGB(255,0,0)
```

Axis.MinorIncrement

Description

Sets or returns the minor unit of measurement of an axis – that is, the gap between its minor gridlines and/or minor ticks marks.

Signature: VB.NET

```
Property MinorIncrement As Double (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the value axis's minor increment to 4
MyChart.ValueAxis.MinorIncrement = 4
```

Axis.MinorTickType

Description

Determines the display of axis tick marks on the minor gridlines.

Signature: VB.NET

```
Property MinorTickType As ASXTickType (read/write)
```

Remarks

Tick marks are marks of measurement on an axis. The values of MinorTickType are:

Invisible Tick	0
Inside Line Tick	1
Outside Line Tick	2
Cross Axis Line Tick	3

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the minor tick type of the value axis
MyChart.ValueAxis.MinorTickType = saxlsOutsideLineTick
```

Axis.MinValue

Description

Sets or returns the minimum value on the value axis.

Signature: VB.NET

```
Property MinValue As Double (read/write)
```

Remarks

> ⚠ MinValue is a property of ValueAxis, and not of CategoryAxis or ZAxis.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the minimum value of the value axis
MyChart.ValueAxis.Minalue = 5
```

Axis.Number

Description

Sets the display format of axis values.

Signature: VB.NET

```
Property Number As Variant (read/write)
```

Remarks

For a complete list of Number codes, see Formatting Codes.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the display format for the axis to 0.00%
MyChart.ValueAxis.Number = 10
```

Axis.TextRotationAngle

Description

Sets or returns the rotation angle of axis values.

Signature: VB.NET

```
Property TextRotationAngle As Integer (read/write)
```

Remarks

Set TextRotationAngle to a value between -90 and 90. Negative values rotate text clockwise, and positive values rotate text counterclockwise.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Rotate the text 45 degrees counterclockwise
MyChart.ValueAxis.TextRotationAngle = 45
```

Axis.Title

Description

Returns a ChartFrame object representing the axis title.

Signature: VB.NET

```
Property Title As IChartFrame (read-only)
```

Remarks

The properties of Title are Area, Border, HasShadow, Height, Text, TextFont, TextHorizontalAlignment, TextRotationAngle, TextVerticalAlignment, Width, X, and Y.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Title returns IChartFrame
MyChart.ValueAxis.Title.Height = 10
```

Axis.ValuesInReverseOrder

Description

Determines whether to display axis values in reverse order.

Signature: VB.NET

```
Property ValuesInReverseOrder As Boolean (read/write)
```

Remarks

When set to True, displays axis values in reverse order.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Display the values in reverse order
MyChart.ValueAxis.ValuesInReverseOrder = True
```

Axis.Wall3D

Description

Returns an Area object representing the walls or floor of a three-dimensional chart.

Signature: VB.NET

```
Property Wall3D As SAArea (read-only)
```

Remarks

CategoryAxis.Wall3D	Returns the walls of a 3-D chart
ValueAxis.Wall3D	Returns the floor of a 3-D chart

The properties of Wall3D are AreaFormatting, BackgroundColor, ForegroundColor, and Pattern.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Format the floor of a 3-D chart
MyChart.ValueAxis.Wall3D.BackgroundColor = RGB(0, 255, 0)
```

Axis.Wall3DBorder

Description

Returns a Line object representing the borders of a 3-D chart's walls or floor.

Signature: VB.NET

```
Property Wall3DBorder As SALine (read-only)
```

Remarks

CategoryAxis.Wall3DBorder	Returns the borders of a 3-D chart's walls
ValueAxis.Wall3DBorder	Returns the borders of a 3-D chart's floor

The properties of Wall3DBorder are LineColor, LineFormatting, LineStyle, and LineWeight.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Format the borders of the floor of a 3-D chart
MyChart.ValueAxis.Wall3DBorder.LineColor = RGB(0, 0, 255)
```

Cell

Description

Represents a single cell in a worksheet.

Signature: VB.NET

```
Public Class Cell
```

Examples

vbscript

```
Set cellB26 = oWorksheet.cells("B26")
cellB26.Value = "Product"
```

Properties

Name	Description
Characters	Returns a Characters object which represents a range of characters within a cell.
Col	Returns the column number of a specified cell.
Format	Sets or returns the format for a cell.
Formula	Sets or returns the formula of a given cell.
Name	Returns the specified cell's name.
Row	Returns the row number of a specified cell.
Style	Sets or returns the Style of a Cell.
Value	Sets or returns the value of a cell

Methods

Name	Description
ClearContents()	Deletes any value or formula within the specified cell.
ClearFormats()	Clears all formatting in the specified cell.
PutValue(Variant)	Puts a value in a cell.

SetAddInFormula(String, String)	SetAddInFormula allows you to use custom formulas from Excel add-in files.

Cell.Characters

Description

Returns a Characters object which represents a range of characters within a cell.

Signature: VB.NET

```
Property Characters(StartIndex As Long, Length As Long) As ICharacters (read/write)
```

Parameters

StartIndex

The character position within the specified cell at which to begin the Characters object.

Length

Number of characters and spaces to include in the Characters object.

Examples

vbscript

```
Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set ws = xlw.Worksheets(1)
strHeader = "Rich Character Formatting Sample"
ws.Cells(3, 3).Value = strHeader
Set oCharacters = ws.Cells(3, 3).Characters(1, Len(strHeader))
oCharacters.Font.Bold = True
oCharacters.Font.Size = 14
```

Cell.ClearContents()

Description

Deletes any value or formula within the specified cell.

Signature: VB.NET

```
Sub ClearContents()
```

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Clear the contents from cell A1
WrkSht.Cells(1, 1).ClearContents
```

Cell.ClearFormats()

Description

Clears all formatting in the specified cell.

Signature: VB.NET

```
Sub ClearFormats()
```

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Clear the formats from cell A1
WrkSht.Cells(1, 1).ClearFormats
```

Cell.Col

Description

Returns the column number of a specified cell.

Signature: VB.NET

```
Property Col As Integer (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, ColNum
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Get the column number of a cell
ColNum = WrkSht.Cells("A1").Col
```

Cell.Format

Description

Sets or returns the format for a cell.

Signature: VB.NET

```
Property Format As SAFormat (read/write)
```

Remarks

Use Format to create a look for a single cell. Both Format and Style define the appearance of cells. However, Style is used to define the look of a set of cells, while Format defines the look of an individual cell.

Define a format for a single cell, rather than create a style, only if it is unlikely that you will apply the same format to other cells. Whenever you define a format for an individual cell, you add format information to the spreadsheet, so if you define the same format repeatedly, the resulting spreadsheet file will be unnecessarily large. When you apply a single Style object to several cells, the style information is written to the spreadsheet only once, conserving space.

The properties of Format are Font, Hidden, HorizontalAlignment, Locked, Number, Orientation, VerticalAlignment, and WrapText. For definitions of each of these properties, see Style.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Make a cell bold-faced with the Format property
WrkSht.Cells(1,1).Format.Font.Bold = True
```

Cell.Formula

Description

Sets or returns the formula of a given cell.

Signature: VB.NET

```
Property Formula As String (read/write)
```

Remarks

Use Formula to add Microsoft Excel formulas to your spreadsheet. For a complete list of available formulas see Formula Functions.

When applying a formula to a cell, always use an equal sign (=), as you would when creating an Excel spreadsheet.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a formula to a cell with the Formula property
WrkSht.Cells(1,1).Value = 1
WrkSht.Cells(2,1).Value = 2
WrkSht.Cells(3,1).Formula = "=SUM(A1:A2)"
```

Cell.Name

Description

Returns the specified cell's name.

Signature: VB.NET

```
Property Name As String (read-only)
```

Remarks

A cell name includes its column letter and row number, for example B5.

Examples

vbscript

```
Dim XlwApp, WrkSht, CellName
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- The Name property returns the Excel address for a cell
CellName = WrkSht.Cells(10, 10).Name '--- returns "J10"
```

Cell.PutValue(Variant)

Description

Puts a value in a cell.

Signature: VB.NET

```
Sub PutValue(newVal As Variant)
```

Parameters

newVal

New Value to put into the cell.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Insert a value into the cell
WrkSht.Cells(1, 1).PutValue "Some value."
```

Cell.Row

Description

Returns the row number of a specified cell.

Signature: VB.NET

```
Property Row As Integer (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, RowNum
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Get the row number of a cell
RowNum = WrkSht.Cells("A1").Row
```

Cell.SetAddInFormula(String, String)

Description

SetAddInFormula allows you to use custom formulas from Excel add-in files.

Signature: VB.NET

```
Sub SetAddInFormula(FileName As String, FormulaVal As String)
```

Parameters

FileName

The name of the file that contains the add-in formula. This parameter can be:

- A complete local or UNC path
- The current directory of the generated spreadsheet
- A sub-directory of the generated spreadsheet's directory
- Up a directory from the current directory of the generated spreadsheet
- Excel's Add-In Library directory or sub-directory

The path specified by FileName is a client path. The path can include forward and/or backward spashes.

FormulaVal

The formula containing the custom function.

Examples

vbscript

```
'--- Pass SetAddInFormula a complete local path:
cell.SetAddInFormula("E:\directory\AddInFile.xla",_
 "=FuncName(A1,A2:A7,...)")

'--- Pass SetAddInFormula a complete UNC path:
cell.SetAddInFormula("\\drive\directory\AddInFile.xla",_
 "=FuncName(A1,A2:A7,...)")

'--- Pass SetAddInFormula the current directory of the spreadsheet or a sub-directory (with or without
a beginning dot):
cell.SetAddInFormula(".\FileInLibraryDir.xla",_
 "=FuncName(A1,A2:A7,...)")

cell.SetAddInFormula("\Dir1\Dir2\FileInLibraryDir.xla",_
 "=FuncName(A1,A2:A7,...)")

'--- Pass SetAddInFormula the directory above the current directory:
cell.SetAddInFormula("..\FileInLibraryDir.xla",_
 "=FuncName(A1,A2:A7,...)")

'--- Pass SetAddInFormula Excel's Add-In Library directory or sub-directory (start with the name of
the file or the sub directory):
cell.SetAddInFormula("FileInLibraryDir.xla",_
 "=FuncName(A1,A2:A7,...)")

cell.SetAddInFormula("SubDir\FileInLibraryDir.xla",_
 "=FuncName(A1,A2:A7,...)")
```

Cell.Style

Description

Sets or returns the Style of a Cell.

Signature: VB.NET

```
Property Style As SAStyle (read/write)
```

Remarks

Use Style to apply a defined Style object to a cell. Both Format and Style define the look of cells, but a format is created for a single cell, while a style is defined for reuse.

Define a format for a single cell, rather than create a style, only if it is unlikely that you will apply the same format to other cells. Whenever you define a format for an individual cell, you add format information to the spreadsheet, so if you define the same format repeatedly, the resulting spreadsheet file will be unnecessarily large. When you apply a single Style object to several cells, the style information is written to the spreadsheet only once, conserving space.

To define a style, first set a Style object, and assign values to any or all of its properties.

Examples

vbscript

```
Dim XlwApp, WrkSht, MyStyle
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Create and configure a SAStyle object
Set MyStyle = XlwApp.CreateStyle()
MyStyle.Font.Name = "Arial"
MyStyle.Font.Bold = True
MyStyle.HorizontalAlignment = sahaRight

'--- Apply the style to a cell
WrkSht.Cells(1, 1).Style = MyStyle
```

Cell.Value

Description

Sets or returns the value of a cell

Signature: VB.NET

```
Property Value As Variant (read/write)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, CellValue
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Put a value into the cell
WrkSht.Cells(1, 1).Value = "Some value."

'--- Read a value from a cell
CellValue = WrkSht.Cells(1, 1).Value
```

Cells

Description

Represents the set of cells in a worksheet.

Signature: VB.NET

```
Public Class Cells
```

Remarks

Address an item of Cells (that is, a single cell), by name or by number.

	Addressing Cells by Name	Addressing Cells by Number
Format	Cells("ColLetterRowNumber")	Cells(row, column)
Example	Cells("D8")	Cells(8, 4)
Advantages	Easy to read, intuitive, matches MS Excel cell references	Faster, easier to use when iterating over a set of cells

Examples

vbscript

```
Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set cells = xlw.Worksheets(1).Cells
```

Properties

Name	Description
Cell	Returns a Cell object.
Columns	Returns specified columns into a Range object.
ColumnWidth	Sets or returns the width of a specified column.
PreserveStrings	Sets or returns whether numeric strings should be inserted as strings or converted to numeric data types.
Range	Returns a range of cells into a Range object
RowHeight	Sets or returns the height of a specified row.
Rows	Returns the specified rows into a Range object.
StandardHeight	Sets or returns the height of all rows where RowHeight was not modified.
StandardWidth	Sets or returns the width of all columns where ColumnWidth was not modified.
SummaryColumn	Sets or returns the location for a Summary Column in relation to the group of columns that comprise the detail block used in an outline.
SummaryRow	Sets the location for a Summary Row in relation to the group of rows that comprise the detail block used in an outline.

UsedRange	Returns a range of used cells within the specified range.

Indexers

Item	Sets or returns a single cell in a Cells collection

Methods

Name	Description
AddHyperlink(Long, Long, Long, Long, String, String())	Inserts a hyperlink in a specified cell.
CopyFromRecordset(RecordSet, Long, Long, Long, Long, Long, String, Long)	Imports a database recordset into a worksheet, fills in headers automatically, and returns the total number of rows imported.
GetCellAt(Long, Long)	Gets the value at a specified cell.
GetDataFromExcelSheet(String, String, Long, Long, Long, Long, Long)	Reads data from an existing spreadsheet.
GetDataFromRecordset(RecordSet, Long, Long, Long, Long, Long, String, SAXLFieldListMode)	Imports a database recordset into a range of cells, fills in field names automatically, and returns the range object.
GroupColumns(Long, Long, Boolean)	Outlines specified columns within a range.
GroupRows(Long, Long, Boolean)	Outlines specified rows within a set of cells.
HPageBreak(SACell)	The page Cells.HPageBreak() does not exist.
MergeCells(Long, Long, Long, Long)	Combines a range of specified cells into a single cell.
VPageBreak(SACell)	Inserts a vertical page break to the right of a specified cell.

Cells.AddHyperlink(Long, Long, Long, Long, String, String())

Description

Inserts a hyperlink in a specified cell.

Signature: VB.NET

```
Sub AddHyperlink( FirstRow As Long, FirstCol As Long, NumRows As Long, NumCols As Long, Address As
String, [SubAddress As String])
```

Parameters

FirstRow

1-based value of the first row of the hyperlink

FirstCol

1-based value of the first column of the hyperlink

NumRows

Number of rows of the hyperlink

NumCols

Number of columns of the hyperlink

Address

URL address of the hyperlink

SubAddress

(optional) Sub address of the hyperlink

Remarks

To insert a hyperlink in a cell, or range of cells,

1. Use AddHyperlink to insert a link in a specified cell, or range of cells. AddHyperlink assigns the upper left cell, the numbers of rows and columns the link will span, an address, and, optionally, a sub-address (e.g. "/search.htm"). The address may be http, https, ftp, mailto, or a UNC path.
2. Merge the range of cells that will contain the link. This is not required. However, note that all cells in the range will function as a single link. Merging the cells in the range makes it clear that the range is a single unit.
3. Assign a value to the cell specified by AddHyperlink. The cell value will function as the link.
4. Apply the hyperlink style to the cell containing the link.

Excel 95 does not support this feature.

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

Cells.AddHyperlink 7, 1, 2, 2, "http://www.softartisans.com"
Cells.MergeCells 7, 1, 2, 2
Cells(7, 1).Value = "SoftArtisans Home Page"
Cells(7, 1).Style = XlwApp.Style("Hyperlink")
```

Cells.Cell

Description

Returns a Cell object.

Signature: VB.NET

```
Property Cell(CellName As String) As SACell (read-only)
```

Examples

$body

Cells.Columns

Description

Returns specified columns into a Range object.

Signature: VB.NET

```
Property Columns(FirstCol As Long, [NumCols As Long = 1]) As SARange (read-only)
```

Parameters

FirstCol

The first column to retrieve

NumCols

(optional)The number of columns to retrieve, including FirstCol. Default value is 1.

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Get a range that holds columns D through F
Set ColRange = Cells.Columns(4, 3)
```

Cells.ColumnWidth

Description

Sets or returns the width of a specified column.

Signature: VB.NET

```
Property ColumnWidth(Column As Long) As Double (read/write)
```

Parameters

Column

The Column for which you wish to set or return the width

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Set column width for the third column to 10
Cells.ColumnWidth(3) = 10

'--- Get the column width for the fifth column
ColWidth = Cells.ColumnWidth(5)
```

Cells.CopyFromRecordset(RecordSet, Long, Long, Long, Long, Long, String, Long)

Description

Imports a database recordset into a worksheet, fills in headers automatically, and returns the total number of rows imported.

Signature: VB.NET

```
Function CopyFromRecordset( _
 RecSet As RecordSet, [ShowFieldNames As Long = 1], _
 [FirstRow As Long = 1], [FirstCol As Long = 1], _
 [MaxRows As Long = 65536], [MaxColumns As Long = 256], _
 [FieldList As String], _
 [FieldListMode As SAXLFieldListMode = saxlsExclude]) As Long
```

Parameters

RecSet

Recordset to copy into the spreadsheet

ShowFieldNames

(optional) Determines whether to import recordset column headers into the first row

FirstRow

(optional) The first row into which to import the recordset

FirstCol

(optional) The first column into which to import the recordset

MaxRows

(optional) Number of rows you wish to bring in

MaxColumns

(optional) Number of columns you wish to bring in

FieldList

(optional) List of fields to include or exclude from the recordset

FieldListMode

(optional) Mode to apply to the FieldList collection – exclude or include

Returns

Total number of rows imported from the recordset.

Remarks

Optional Parameter	Default Value	
ShowFieldNames	**True**	Recordset column headers will be imported to the first row
FirstRow	**1**	The recordset will be imported into cell A1
FirstColumn	**1**	
MaxRows	**65536**	Excel 95 will delete rows after 16,384
MaxColumns	**256**	
FieldList	**""**	All fields in the recordset are used
FieldListInclude	**saxlsExclude**	Excludes fields listed in FieldList parameter

> FieldList parameter is a comma delimited string specifying recordset fields to include or exclude. They may be referenced by field name or position, or any combination. The order fields are spcified in the FieldList parameter is the order in which they will return.

Examples

vbscript

```
Set oConn=  Server.CreateObject("ADODB.Connection")
oConn.Open "DBQ="&filePath& ";Driver = {MicrosoftAccessDriver(*.mdb)};" & _
 "DriverId = 25;FIL = MSAccess;"
Set oRs = oConn.Execute("select * from qryOrders")

Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set cells = xlw.Worksheets(1).Cells
'--- CopyFromRecordset "oRs", display recordset headers,
'--- populate spreadsheet beginning with cell "A1" for a
'--- maximum of 100 rows and 100 columns, include
'--- only columns 1, 2, and 3 of the recordset.
NumRows = Cells.CopyFromRecordset(oRs, True, 1, 1, 100, 100, "1, 2, 3",_
    saxlsInclude)

xlw.Save "c:\workbook2.xls"

Set xlw = nothing
```

Cells.GetCellAt(Long, Long)

Description

Gets the value at a specified cell.

Signature: VB.NET

```
Function GetCellAt(Row As Long, Column As Long) As SACell
```

Parameters

Row

Row coordinate of the cell to return

Column

Column coordinate of the cell to return

Returns

Cell object representing the specified cell.

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Get cell B6
Set CellB6 = Cells.GetCellAt(6, 2)
```

Cells.GetDataFromExcelSheet(String, String, Long, Long, Long, Long, Long)

Description

Reads data from an existing spreadsheet.

Signature: VB.NET

```
Function GetDataFromExcelSheet(FileName As String, [Source As String], _
 [IncludeFormatting As Long = 1], [FirstRow As Long = 1], _
 [FirstCol As Long = 1], [MaxRows As Long = 65536], _
 [MaxCols As Long = 256]) As SARange
```

Parameters

FileName

Full path and name of the Excel spreadsheet from which to import

Source

(optional) String representing the range of cells from which to draw the data from the spreadsheet

IncludeFormatting

(optional) Whether or not to preserve formatting from the source spreadsheet

FirstRow

(optional) First row of the destination spreadsheet into which to import data

FirstCol

(optional) First column of the destination spreadsheet into which to import data

MaxRows

(optional) Number of rows of data to import

MaxCols

(optional) Number of columns of data to import

Returns

Range object representing the specified range of cells from the worksheet.

Remarks

FileName is the name of the source file, that is, the spreadsheet from which ExcelWriter is reading. The optional parameter IncludeFormatting specifies whether to copy formatting from the source sheet. The optional parameters FirstRow, FirstCol, MaxRows, and MaxCols specify where to write the data in the destination spreadsheet.

The optional parameter Source specifies the first cell in the source range of cells. Source may include both sheet number and cell name (e.g. **Sheet2!$B$11**), or a cell name alone (e.g. "B11"). If Source does not include a sheet reference, ExcelWriter will read from the first sheet in the workbook. If you do not assign a Source value, ExcelWriter will start reading at cell A1 in the first worksheet.

Optional Parameter	**Default Value**	
Source	**""**	If you do not assign a Source value, ExcelWriter will start reading at cell A1 in the first worksheet.
IncludeFormatting	**1**	Limitation: ExcelWriter will not include formatting in merged cells.
FirstRow	**1**	
FirstColumn	**1**	
MaxRows	**16384 in Excel 95 (BIFF7); 65536 in Excel 97/2000 (BIFF8)**	
MaxColumns	**256**	(Limit imposed by Microsoft's BIFF8 format, not by ExcelWriter)

Examples

vbscript

```
Set xlw = Server.CreateObject("Softartisans.Excelwriter")
Set ws = xlw.Worksheets(1)

'--- Get data from workbook1.xls, sheet 1, starting at cell B3,
'--- with formatting, and copy to cells B3:F12 in the destination
'--- file.
Set ResultRange = ws.Cells.GetDataFromExcelSheet("C:\spreadsheets\workbook1.xls", "B3",_
     True, 3, 2, 10, 5)

'--- Save the destination file
xlw.Save "c:\workbook2.xls"

Set xlw = Nothing
```

Cells.GetDataFromRecordset(RecordSet, Long, Long, Long, Long, Long, String, SAXLFieldListMode)

Description

Imports a database recordset into a range of cells, fills in field names automatically, and returns the range object.

Signature: VB.NET

```
Function GetDataFromRecordset(RecordSet As RecordSet, _
[ShowFieldNames As Long = 1], _
[FirstRow As Long = 1], _
[FirstCol As Long = 1], _
[MaxRows As Long = 65536], _
[MaxColumns As Long = 256], _
[FieldList As String], _
[Mode As SAXLFieldListMode = saxlsExclude]) As SARange
```

Parameters

RecordSet

RecordSet to import

ShowFieldNames

(optional) Whether or not to show the recordset field names in the first row

FirstRow

(optional) First row into which to import data

FirstCol

(optional) First column into which to import data

MaxRows

(optional) Number of rows to import

MaxColumns

(optional) Number of columns to import

FieldList

(optional) List of fields to include or exclude

Mode

(optional) Dictate the behavior of the FieldList: include or exclude

Returns

Range object representing the cells filled using the current method

Remarks

ShowFieldNames, FirstRow, FirstColumn, MaxRows, MaxColumns, FieldList, and FieldListInclude are optional. The following table lists their default values.

Optional Parameter	Default Value	
ShowFieldNames	**True**	Recordset column headers will be imported to the first row
FirstRow	**1**	The recordset will be imported into cell A1
FirstColumn	**1**	
MaxRows	**65536**	Excel 95 will delete rows after 16,384
MaxColumns	**256**	
FieldList	**""**	All fields in the recordset are used
FieldListInclude	**saxlsExclude**	Excludes fields listed in FieldList parameter

FieldList parameter is a comma delimited string specifying recordset fields to include or exclude. They may be referenced by field name or position, or any combination. The order fields are spcified in the FieldList parameter is the order in which they will return.

Examples

vbscript

```
Set oConn = Server.CreateObject("ADODB.Connection")
oConn.Open "DBQ= " & filePath & ";Driver = {MicrosoftAccessDriver(*.mdb)};" & _
   "DriverId = 25;FIL = MSAccess;"
Set oRs = oConn.Execute("select * from qryOrders")

Set xlw = Server.CreateObject("Softartisans.ExcelWriter")
Set cells = xlw.Worksheets(1).Cells

'--- GetDataFromRecordset "oRs", display recordset headers,
'--- populate spreadsheet beginning  with cell "A1" for a
'--- maximum of 100 rows and 100 columns, include only columns
'--- 1, 2, and 3 of the recordset.
NumRows = Cells.GetDataFromRecordset(oRs, True, 1, 1, 100, 100, "1, 2, 3", _
   saxlsInclude)

xlw.Save "c:\workbook2.xls"

Set xlw = Nothing
```

Cells.GroupColumns(Long, Long, Boolean)

Description

Outlines specified columns within a range.

Signature: VB.NET

```
Sub GroupColumns(FirstColumn As Long, _
[NumColumns As Long = 1], _
[Collapsed As Boolean = False])
```

Parameters

FirstColumn

First column to be included in the outline. The first column can be designated as a summary column, (see the Cells.SummaryColumn property) and, if so, should be visible. Columns that follow or precede the summary column contain the outline detail and may be visible or hidden, depending on the value of the collapsed parameter. Column numbering is 1-based.

NumColumns

(Optional) Default value: 1. 1-based number of columns to import.

Collapsed

(Optional) Default value: False. If set to True, the outline detail columns will be hidden. If set to False, the outline detail columns will be visible.

Remarks

> If you wish to designate the location of the summary column as the last column of the outlined group, then the Cells.SummaryColumn property must be set using the saxlSummaryOnRight (0) value. The summary column should always be visible.

Examples

vbscript

```
'--- Outline columns A to D, and hide the outline detail columns.
'--- Column A is the summary column and will be visible. This
'--- assumes that you have set the "SummaryColumn" property to
'--- place summary columns to the left of detail columns. Columns
'--- B to D are the outline detail columns and will be hidden.
cells.GroupColumns 1, 4, True
```

See also: Cells.GroupRows(Long, Long, Boolean)

Cells.GroupRows(Long, Long, Boolean)

Description

Outlines specified rows within a set of cells.

Signature: VB.NET

```
Sub GroupRows(FirstRow As Long, _
     [NumRows As Long = 1], _
     [Collapsed As Boolean = False])
```

Parameters

FirstRow

First row to be included in the outline. The first row can be designated as a summary row, (see the Cells.SummaryRow property) and, if so, should be visible. Rows that follow or precede the summary row contain the outline detail and may be visible or hidden, depending on the value of the collapsed parameter. Row numbering is 1-based.

NumRows

(Optional) Default value: 1. 1-based number of rows to group.

Collapsed

(Optional) Default value: False. If set to True, the outline detail rows will be hidden. If set to False, the outline detail rows will be visible.

Remarks

If you wish to designate the location of the summary row as the last row of the outlined group, then the Cells.SummaryRow property must be set using the saxlSummaryBelow (0) value. The summary row should always be visible.

Examples

vbscript

```
'--- Outline rows 6 to 15, and hide the outline detail rows.
'--- Row 6 is the summary row and will be visible. This assumes
'--- that you have set the "SummaryRow" property to place summary
'--- row above the detail rows. Rows 7 and 15 are the outline
'--- detail rows and will be hidden.
cells.GroupRows 6, 10, True
```

See also: Cells.GroupColumns(Long, Long, Boolean)

Cells.HPageBreak(SACell)

Description

Inserts a horizontal page break after a specified cell.

Signature: VB.NET

```
Sub HPageBreak(Cell As SACell)
```

Parameters

Cell

Cell after which to insert a horizontal page break

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Create a horizontal page break after cell C1
Cells.HPageBreak Cells(1, 3)
```

Cells.Item

Description

Sets or returns a single cell in a Cells collection

Signature: VB.NET

```
Property Item(RowOrName, [Col As Long]) As SACell (read/write)
```

Parameters

RowOrName

If a string, the name of the cell ("A1"). If a Long, the row in which the cell is located.

Col

(optional) The column in which the cell is located. Use only when setting RowOrName equal to a Long.

Remarks

In VBScript, this is the default property for the Cells object.

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Use the default Items property to access each
'--- cell object in the cells collection

'--- Get a reference to cell A1
Set CellA1 = Cells("A1") ' Cells(1,1) also works

'--- Set some properties for cell B2
Cells(2, 2).Value = "This is cell B2"
Cells("B2").Format.Font.Bold = True
```

Cells.MergeCells(Long, Long, Long, Long)

Description

Combines a range of specified cells into a single cell.

Signature: VB.NET

```
Sub MergeCells(FirstRow As Long, FirstCol As Long, _
     NumRows As Long, NumCols As Long)
```

Parameters

FirstRow

First row of the cells to merge

FirstCol

First column of the cells to merge

NumRows

Number of rows to merge

NumCols

Number of columns to merge

Remarks

Use the address of the upper-left cell in the range to reference the merged cell.

> ⚠ Excel 95 does not support this feature.

Examples

vbscript

```
Set Xlwapp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Merge cells A10:C11
Cells.MergeCells 10, 1, 2, 3
Cells(10, 1).Value = "Merged cells."
```

Cells.PreserveStrings

Description

Sets or returns whether numeric strings should be inserted as strings or converted to numeric data types.

Signature: VB.NET

```
Property PreserveStrings As Boolean (read/write)
```

Remarks

PreserveStrings is set to False by default. When PreserveStrings is set to False, numeric cell entries are interpreted as numbers and right-aligned. If you set PreserveStrings to True, numbers are preserved as strings and, therefore, left-aligned.

Cells.Range

Description

Returns a range of cells into a Range object

Signature: VB.NET

```
Property Range([FirstRow As Long = 1], [FirstCol As Long = 1], [NumRows As Long = 65536], [NumCols As
Long = 256]) As SARange (read-only)
```

Parameters

FirstRow

(optional) First row to return. Default value is 1.

FirstCol

(optional) First column to return. Default value is 1.

NumRows

(optional) Number of rows to return. Default value is 65536 (all rows)

NumCols

(optional) Number of columns to return. Default value is 256 (all columns)

Remarks

By default, Range retrieves all cells in a Cells object.

Examples

vbscript

```
Set Xlwapp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Get a range of cells B5:E9
Set MyRange = Cells.Range(5, 2, 5, 4)
```

Cells.RowHeight

Description

Sets or returns the height of a specified row.

Signature: VB.NET

```
Property RowHeight(Row As Long) As Double (read/write)
```

Parameters

Row

Row for which you wish to return or set the height.

Examples

vbscript

```
Set Xlwapp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Set row 6 to a height of 10
Cells.RowHeight(6) = 10

'--- Get the height of row 9
RowHeight = Cells.RowHeight(9)
```

Cells.Rows

Description

Returns the specified rows into a Range object.

Signature: VB.NET

```
Property Rows(FirstRow As Long, [NumRows As Long = 1]) As SARange
```

Parameters

FirstRow

First Row to return

NumRows

(optional) Number of rows to return. Default value is 1.

Examples

vbscript

```
Set Xlwapp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Get a range that holds rows 9-12
Set RowRange = Cells.Rows(9, 4)
```

Cells.StandardHeight

Description

Sets or returns the height of all rows where RowHeight was not modified.

Signature: VB.NET

```
Property StandardHeight As Double (read/write)
```

Examples

vbscript

```
Set Xlwapp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Set StandardHeight to 5
Cells.StandardHeignt = 5

'--- StandardHeight is a read/write property
'--- The value may also be read
Dim CellHeight
CellWidth = Cells.StandardHeight
```

Cells.StandardWidth

Description

Sets or returns the width of all columns where ColumnWidth was not modified.

Signature: VB.NET

```
Property StandardWidth As Double (read/write)
```

Examples

vbscript

```
Set Xlwapp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Set StandardWidth to 20
Cells.StandardWidth = 20

'--- StandardWidth is a read/write property
'--- The value may also be read
Dim CellWidth
CellWidth = Cells.StandardWidth
```

Cells.SummaryColumn

Description

Sets or returns the location for a Summary Column in relation to the group of columns that comprise the detail block used in an outline.

Signature: VB.NET

```
Property SummaryColumn As SAXlSummaryColumn (read/write)
```

Remarks

This property is used in association with the GroupColumns method that sets an outline level. It should be set prior to calling this method. Possible values are saxlSummaryOnLeft (-1), the default, or saxlSummaryOnRight (0).

> When designing a report that will contain summary columns, be sure to allow for an empty column to contain the summary column. When using the setting, saxlSummaryOnLeft (-1), the empty column must precede the detail columns, and when using saxlSummaryOnRight (0), it must follow the detail columns.

Examples

vbscript

```
Set Xlwapp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Set SummaryColumn before calling GroupColumns
Cells.SummaryColumn = saxlSummaryOnRight

'--- Group colums A through D
Cells.GroupColumns 1, 4, True
```

Cells.SummaryRow

Description

Sets the location for a Summary Row in relation to the group of rows that comprise the detail block used in an outline.

Signature: VB.NET

```
Property SummaryRow As SAXlSummaryRow (read/write)
```

Remarks

This property is used in association with the GroupRows method that sets an outline level. It should be set prior to calling this method. Possible values are saxlSummaryAbove (-1) – the default – or saxlSummaryBelow (0).

> When designing a report that will contain summary rows, be sure to allow for an empty row to contain the summary row. When using the setting, saxlSummaryAbove (-1), the empty row must precede the detail rows, and when using saxlSummaryBelow (0), it must follow the detail rows.

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Set SummaryRow before calling GroupRows
Cells.SummaryRow = saxlSummaryAbove

'--- Group rows 6 through 15
Cells.GroupRows 6, 10, True
```

Cells.UsedRange

Description

Returns a range of used cells within the specified range.

Signature: VB.NET

```
Property UsedRange As SARange (read-only)
```

Remarks

If there are no use cells in the range, an empty range is returned.

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange, UsedRng
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- This range is A1:B6
Set MyRange = WrkSht.Cells.Range(1, 1, 6, 2)

'--- Get the range of used cells
Set UsedRng = MyRange.UsedRange
```

Cells.VPageBreak(SACell)

Description

Inserts a vertical page break to the right of a specified cell.

Signature: VB.NET

```
Sub VPageBreak(Cell As SACell)
```

Parameters

Cell

Cell after which to insert the vertical page break

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Set a VPageBreak after cell J10
Cells.VPageBreak Cells(10, 10)
```

Characters

Description

Represents a range of characters within a cell.

Signature: VB.NET

```
Public Class Characters
```

Remarks

The Characters object is accessible through the property Cell.Characters.

Properties

Name	Description
Font	Sets or returns the Font object used to format text within the specified Characters object.
Index	Returns the start index (the position of the first character) of the specified Characters object.
Length	Returns the length (number of characters and spaces) in the specified Characters object.

Characters.Font

Description

Sets or returns the Font object used to format text within the specified Characters object.

Signature: VB.NET

```
Property Font As SAFont (read/write)
```

Examples

vbscript

```
Set xlw = Server.CreateObject("Softartisans.excelwriter")
Set ws = xlw.Worksheets(1)
strHeader = "Rich Character Formatting Sample"
ws.Cells(3, 3).Value = strHeader
Set oCharacters = ws.Cells(3, 3).Characters(1, Len(strHeader))
oCharacters.Font.Bold = True
oCharacters.Font.Size = 14
```

Characters.Index

Description

Returns the start index (the position of the first character) of the specified Characters object.

Signature: VB.NET

```
Property Index As Long (read-only)
```

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
```

Characters.Length

Description

Returns the length (number of characters and spaces) in the specified Characters object.

Signature: VB.NET

```
Property Length As Long (read-only)
```

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
```

Chart

Description

Represents a single chart in a worksheet.

Signature: VB.NET

```
Public Class Chart
```

Remarks

To create a Chart object, use Charts.Add(ASXChartType, Integer, Long, Long, Long, Long).

Examples

vbscript

```
Set Chart = charts.Add(1, 3, 3, 0, 15, 7)
```

Properties

Name	Description

AutoScale	Sets or returns whether Excel will automatically re-size a chart's text when the chart is re-sized.
BopPopGapSize	Sets or returns the size of the gap between the two pies in a pie of pie chart or the pie and bar in a bar of pie chart.
BopPopSecondPieSize	Sets or returns the size of the second pie in 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.
CategoryAxis	Returns an Axis object representing the chart's X axis.
ChartArea	Returns a ChartFrame object representing the chart area in the worksheet.
DataTable	Returns a ChartFrame object representing a chart's data table.
DataTableHasHorizontalBorders	Sets or returns whether the chart's data table will display horizontal grid lines.
DataTableHasVerticalBorders	Sets or returns whether the chart's data table will display vertical grid lines
DataTableShowsSeriesKeys	Sets or returns whether the chart's data table will include legend keys next to the series titles.
FillSurfaceArea	Sets or returns whether the surface area of a Surface chart (type 5) will be filled or appear transparent
Legend	Returns a ChartFrame object representing the chart legend.
LegendLocation	Sets or returns the position of the legend in relation to the chart's plot area
LegendVertical	Sets or returns whether the legend will be displayed vertically.
PlotArea	Returns a ChartFrame object representing the chart's plot area – that is, the area of the chart itself.
RectangularCorner	Sets or returns whether the chart area will be sharp-cornered or round-cornered.
SecondaryCategoryAxis	Returns an Axis object representing a chart's secondary X axis.
SecondaryValueAxis	Returns an Axis object representing the chart's secondary Y axis.

SeriesCollection	Returns a SeriesCollection object representing the set of data series in the chart.
ShowDataTable	Sets or returns whether a data table is displayed in place of the category axis tick mark labels in some chart types.
ShowHorizontal	Sets or returns whether columns are displayed horizontally.
ShowLegend	Sets or returns whether the chart legend is displayed
ShowValueAsPercent	Sets or returns whether the Value Axis values are displayed as percents
SizeOfCenterHole	Sets or retrieves the size of the center hole of a doughnut chart.
SliceDistanceFromCenter	Sets or returns the distance of slices from the center of an exploded pie chart
SpaceBetweenBars	Sets or retrieves the space between data series bars.
SpaceBetweenCategories	Sets or retrieves the space between categories in column and bar charts.
StackDisplayedValues	Sets or returns whether series values will be stacked.
Subtype	Returns the chart's subtype code.
Title	Returns a ChartFrame object representing the chart's title.
Type	Returns the chart's type code.
Valueaxis	Returns an Axis object representing the chart's Y axis.
View3D	Returns a 3DProperties object representing the properties of a three-dimensional chart.

Methods

Name	Description
Move(Long, Long, Long, Long)	Moves the chart to a specified location.
SetChartType(ASXChartType, Integer)	Sets the chart's type and subtype.

Chart.AutoScale

Description

Sets or returns whether Excel will automatically re-size a chart's text when the chart is re-sized.

Signature: VB.NET

```
Property AutoScale As Boolean (read/write)
```

Remarks

If AutoScale is set to True, Excel will automatically resize a chart's text when the chart is resized. If AutoScale is set to False, Excel will not automatically scale a chart's text.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 2, 2, 10, 10)

'--- Enable AutoScale for the chart
MyChart.AutoScale = True
```

Chart.BopPopGapSize

Description

Sets or returns the size of the gap between the two pies in a pie of pie chart or the pie and bar in a bar of pie chart.

Signature: VB.NET

```
Property BopPopGapSize As Integer (read/write)
```

Remarks

By default, BopPopGapSize is set to 50% of the width of the second pie, or of the bar.

> ⚠ Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new Pie Of Pie chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsPieChart, 2, 2, 2, 10, 10)

'--- Set BopPopGapSize to 100% the width of the second pie
MyChart.BopPopGapSize = 100
```

Chart.BopPopSecondPieSize

Description

Sets or returns the size of the second pie in a pie of pie chart.

Signature: VB.NET

```
Property BopPopSecondPieSize As Integer (read/write)
```

Remarks

By default, BopPopSecondPieSize is set to 50% of the first pie.

> Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new Pie Of Pie chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsPieChart, 2, 2, 2, 10, 10)

'--- Set BopPopSecondPieSize to 75% of the size of the first pie
MyChart.BopPopSecondPieSize = 75
```

Chart.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: VB.NET

```
Property BopPopSplitType As ASXBopPopSplitType (read/write)
```

Remarks

Use BopPopSplitType in conjunction with BopPopSplitValue. Assign BopPopSplitType by name or number, according to the following table.

Name	Number	Definition
saxlsBopPopPosition	**0**	The bar or second pie will contain the chart's last n data points, where n is the value assigned to BopPopSplitValue.
saxlsBopPopValue	**1**	The bar or second pie will contain all data points with values less than n, where n is the value assigned to BopPopSplitValue.
saxlsBopPopPercent	**2**	The bar or second pie will contain data points with values less than n% of the total data series in the chart, where n is the value assigned to BopPopSplitValue.
saxlsBopPopCustom	**3**	The bar or second pie is customizable. To customize, drag data points between plot areas.

> Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new Pie Of Pie chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsPieChart, 2, 2, 2, 10, 10)

'--- Set BopPopSplitType to saxlsBopPopPercent
MyChart.BopPopSplitType = saxlsBopPopPercent
MyChart.BopPopSplitValue = 40
```

Chart.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.

Signature: VB.NET

```
Property BopPopSplitValue As Integer (read/write)
```

Remarks

BopPopSplitValue represents a data point value, data point position, or data point percentage, depending on the value of BopPopSplitType.

If BopPopSplitType = BopPopSplitValue represents

saxlsBopPopPosition	The number of last data points that will be included in the bar or second pie. For example, if BopPopSplitValue is set to 3, the bar or second pie will contain the last 3 data points in the pie's data series.
saxlsBopPopValue	The value of a data point. Only data points with values less than this value will be included in the bar or second pie. For example, if BopPopSplitValue is set to 15, the bar or second pie will contain all data points with values less than 15.
saxlsBopPopPercent	A data point's percentage value. Only data points with a percentage less than this value will be included in the bar or second pie. For example, if BopPopSplitValue is set to 40, the bar or second pie will contain all data points with percentage values less than 40%.

Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new Pie Of Pie chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsPieChart, 2, 2, 2, 10, 10)

'--- Set BopPopSplitType to saxlsBopPopPercent
MyChart.BopPopSplitType = saxlsBopPopPercent
MyChart.BopPopSplitValue = 40
```

Chart.CategoryAxis

Description

Returns an Axis object representing the chart's X axis.

Signature: VB.NET

```
Property CategoryAxis As IAxis (read-only)
```

Remarks

The properties of CategoryAxis are AxisCrossesMidCategory, AxisLine, LabelFont, LabelPosition, MajorGridline, MajorIncrement, MajorTickType, MinorGridline, MinorIncrement, MinorTickType, Number, Title, ValuesInReverseOrder, Wall3D, and Wall3DBorder.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new bar chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsBarChart, 0, 2, 2, 10, 10)

'--- The CategoryAxis property is an IAxis interface that you can
'--- use to adjust the properties of the chart's category axis
'--- Set the axis title
MyChart.CategoryAxis.Title = "Category Axis Title"
```

Chart.ChartArea

Description

Returns a ChartFrame object representing the chart area in the worksheet.

Signature: VB.NET

```
Property ChartArea As IChartFrame (read-only)
```

Remarks

ChartArea is the area defined by Charts.Add(ASXChartType, Integer, Long, Long, Long, Long) – that is, the entire area within the worksheet assigned to the chart. The properties of ChartArea are Area, Border, HasShadow, Height, Text, TextFont, TextHorizontalAlignment, TextRotationAngle, TextVerticalAlignment, Width, X, and Y.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new bar chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsBarChart, 0, 2, 2, 10, 10)

'--- The ChartArea property is an IChartFrame interface that you can
'--- use to adjust the properties of the chart's frame
'--- Adjust the height and width of the chart's frame
'--- Apply shadowing to the border
MyChart.ChartArea.HasShadow = True
MyChart.ChartArea.Width = 10
MyChart.ChartArea.Height = 10
```

Chart.DataTable

Description

Returns a ChartFrame object representing a chart's data table.

Signature: VB.NET

```
Property DataTable As IChartFrame (read-only)
```

Remarks

The properties of DataTable are Area, Border, HasShadow, Height, Text, TextFont, TextHorizontalAlignment, TextRotationAngle, TextVerticalAlignment, Width, X, and Y.

> ⚠ Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new bar chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsBarChart, 0, 2, 2, 10, 10)

'--- First, enable the data table for the chart
MyChart.ShowDataTable = True

'--- The DataTable property is an IChartFrame interface that you can
'--- use to adjust the properties of the chart's data table
'--- Adjust the height and width of the data table
'--- Apply shadowing to the border
MyChart.DataTable.HasShadow = True
MyChart.DataTable.Width = 10
MyChart.DataTable.Height = 10
```

See also: ShowDataTable.

Chart.DataTableHasHorizontalBorders

Description

Sets or returns whether the chart's data table will display horizontal grid lines.

Signature: VB.NET

```
Property DataTableHasHorizontalBorders As Boolean (read/write)
```

Remarks

When set to True, the chart's data table will display horizontal grid lines. DataTableHasHorizontalBorders is set to True by default.

> ⚠ Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new bar chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsBarChart, 0, 2, 2, 10, 10)

'--- First, show the data tabke
MyChart.ShowDataTable = True

'--- Set DataTableHasHorizontalBorders to False to hide
'--- horizontal borders in the data table
MyChart.DataTableHasHorizontalBorders = False
```

See also: ShowDataTable.

Chart.DataTableHasVerticalBorders

Description

Sets or returns whether the chart's data table will display vertical grid lines

Signature: VB.NET

```
Property DataTableHasVerticalBorders As Boolean (read/write)
```

Remarks

When set to True, the chart's data table will display vertical grid lines. DataTableHasVerticalBorders is set to True by default.

> ⚠ Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new bar chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsBarChart, 0, 2, 2, 10, 10)

'--- First, show the data tabke
MyChart.ShowDataTable = True

'--- Set DataTableHasVerticalBorders to False to hide
'--- vertical borders in the data table
MyChart.DataTableHasVerticalBorders = False
```

See also: ShowDataTable.

Chart.DataTableShowsSeriesKeys

Description

Sets or returns whether the chart's data table will include legend keys next to the series titles.

Signature: VB.NET

```
Property DataTableShowsSeriesKeys As Boolean (read/write)
```

Remarks

When set to True, the chart's data table will include legend keys next to the series titles. DataTableShowsSeriesKeys is set to True by default.

> ⚠ Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new bar chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsBarChart, 0, 2, 2, 10, 10)

'--- First, show the data tabke
MyChart.ShowDataTable = True

'--- Set DataTableShowsSeriesKeys to False to hide
'--- legend keys next to the series titles
MyChart.DataTableShowsSeriesKeys = False
```

See also: ShowDataTable.

Chart.FillSurfaceArea

Description

Sets or returns whether the surface area of a Surface chart (type 5) will be filled or appear transparent

Signature: VB.NET

```
Property FillSurfaceArea As Long (read/write)
```

Remarks

FillSurfaceArea is set to 1 (enabled) by default. To display a transparent surface area, set FillSurfaceArea to 0.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new surface chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsSurfaceChart, 0, 2, 2, 10, 10)

'--- For a surface chart, set FillSurfaceArea to 0 to get
'--- a transparent surface area
MyChart.FillSurfaceArea = 0
```

Chart.Legend

Description

Returns a ChartFrame object representing the chart legend.

Signature: VB.NET

```
Property Title As IChartFrame (read-only)
```

Remarks

The properties of Legend are Area, Border, HasShadow, Height, Text, TextFont, TextHorizontalAlignment, TextRotationAngle, TextVerticalAlignment, Width, X, and Y.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new surface chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsSurfaceChart, 0, 2, 2, 10, 10)

'--- The Legend property returns a IChartFrame interface that
'--- you can use to adjust the properties of the legend
'--- Set the height and width of the chart's legend
MyChart.Legend.Height = 10
MyChart.Legend.Width = 10
```

Chart.LegendLocation

Description

Sets or returns the position of the legend in relation to the chart's plot area

Signature: VB.NET

```
Property LegendLocation As ASXObjectLocation (read/write)
```

Remarks

Determines the position of the legend in relation to the chart's plot area. LegendLocation has six possible values. Assign LegendLocation by name or number according to the following table.

saxlsObjectBottom	0
saxlsObjectCorner	1
saxlsObjectTop	2
saxlsObjectRight (default)	3
saxlsObjectLeft	4
saxlsObjectNotDocked	7

> LegendLocation and the pair of Chartframe properties Height and Width are mutually exclusive. Use either LegendLocation or Height and Width.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new surface chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsSurfaceChart, 0, 2, 2, 10, 10)

'--- Set the location of the chart's legend to the left
MyChart.LegendLocation = saxlsObjectLeft
```

Chart.LegendVertical

Description

Sets or returns whether the legend will be displayed vertically.

Signature: VB.NET

```
Property LegendVertical As Boolean (read/write)
```

Remarks

Displays the legend vertically when set to True. LegendVertical is set to True by default. To display the legend horizontally, set LegendVertical to False.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 2, 2, 10, 20)

'--- LegendVertical is True by default.
'--- Set it to False to display the legend horizontally
MyChart.LegendVertical = False
```

Chart.Move(Long, Long, Long, Long)

Description

Moves the chart to a specified location.

Signature: VB.NET

```
Sub Move(UpperLeftColumn As Long, _
     UpperLeftRow As Long, _
     LowerRightColumn As Long, _
     LowerRightRow As Long)
```

Parameters

UpperLeftColumn

Index of the column of the upper left corner of the destination of the chart

UpperLeftRow

Index of the row of the upper left corner of the destination of the chart

LowerRightColumn

Index of the column of the lower right corner of the destination of the chart

LowerRightRow

Index of the row of the lower right corner of the destination of the chart

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- You can use the Move method to move or resize a chart
'--- to a new location in the worksheet
'--- Move the chart's upper left corner from
'--- cell B2 to cell D10 (col 4, row 10)
MyChart.Move 4, 10, 30, 30
```

Chart.PlotArea

Description

Returns a ChartFrame object representing the chart's plot area – that is, the area of the chart itself.

Signature: VB.NET

```
Property PlotArea As IChartFrame (read-only)
```

Remarks

The properties of PlotArea are Area, Border, HasShadow, Height, Text, TextFont, TextHorizontalAlignment, TextRotationAngle, TextVerticalAlignment, Width, X, and Y.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- The PlotArea property returns an IChartFrame interface that
'--- you can use to adjust the properties of the chart's plot area
'--- Set the height, width, and shadowing of the plot area
MyChart.PlotArea.Height = 10
MyChart.PlotArea.Width = 10
MyChart.PlotArea.HasShadow = True
```

Chart.RectangularCorner

Description

Sets or returns whether the chart area will be sharp-cornered or round-cornered.

Signature: VB.NET

```
Property RectangularCorner As Long (read/write)
```

Remarks

Determines whether the chart area will be sharp-cornered or round-cornered. By default, RectangularCorner is set to 0, and chart area corners are angular. For round corners, set RectangularCorner to 1.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set RectangularCorner to 1 to get round corners on the chart
MyChart.RectangularCorner = 1
```

Chart.SecondaryCategoryAxis

Description

Returns an Axis object representing a chart's secondary X axis.

Signature: VB.NET

```
Property SecondaryCategoryAxis As IAxis (read-only)
```

Remarks

The properties of SecondaryCategoryAxis are AxisCrossesMidCategory, AxisLine, LabelFont, LabelPosition, MajorGridline, MajorIncrement, MajorTickType, MinorGridline, MinorIncrement, MinorTickType, Number, Title, ValuesInReverseOrder, Wall3D, and Wall3DBorder.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and create a new 2 axis chart
Dim XlwApp, WrkSht, MyChart, SecondAxis
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnLine2AxesChart, _
   0, 2, 2, 10, 20)

'--- SecondaryCategoryAxis returns an IAxis interface
'--- for the second category axis of a saxlsColumnLine2AxesChart
Set SecondAxis = MyChart.SecondaryCategoryAxis
```

Chart.SecondaryValueAxis

Description

Returns an Axis object representing the chart's secondary Y axis.

Signature: VB.NET

```
Property SecondaryValueAxis As IAxis (read-only)
```

Remarks

The properties of SecondaryValueAxis are AxisLine, CategoryAxisCrossMaxValue, CategoryCrossingValue, LabelFont, LabelPosition, LogarithmicScale, MajorGridline, MajorIncrement, MajorTickType, MaxValue, MinorGridline, MinorIncrement, MinorTickType, MinValue, Number, Title, ValuesInReverseOrder, Wall3D, and Wall3DBorder.

By default, for a chart's secondary value axis CategoryAxisCrossMaxValue is set to true and the secondary value axis will cross the category axis at the maximum category value.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and create a new 2 axis chart
Dim XlwApp, WrkSht, MyChart, SecondValueAxis
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnLine2AxesChart, _
   0, 2, 2, 10, 20)

'--- SecondaryValueAxis returns an IAxis interface
'--- for the second value axis of a saxlsColumnLine2AxesChart
Set SecondValueAxis = MyChart.SecondaryValueAxis
```

Chart.SeriesCollection

Description

Returns a SeriesCollection object representing the set of data series in the chart.

Signature: VB.NET

```
Property SeriesCollection As ISeriesCollection (read-only)
```

Remarks

The methods and properties of SeriesCollection are Add, CategoryData, and Item.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart, MySeries
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- SeriesCollection returns an ISeriesCollection interface
'--- After creating a chart, use the SeriesCollection property
'--- to add one or more data series
'--- Chart's data will be based on values in A1:B10
Set MySeries = MyChart.SeriesCollection.Add("A1:B10", , 1)

'--- Chart's Category Data (Y axis) will be based on D1:D2
MyChart.SeriesCollection.CategoryData = "D1:D2"
```

Chart.SetChartType(ASXChartType, Integer)

Description

Sets the chart's type and subtype.

Signature: VB.NET

```
Sub SetChartType(Type As ASXChartType, Subtype As Integer)
```

Parameters

Type

New type of the chart

SubType

New SubType of the chart

Remarks

Consult Chart Codes to select ChartType (pie, column, etc.) and SubType.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Convert clustered column chart to a 3D Pie chart
MyChart.SetChartType saxlsPieChart, 1
```

Chart.ShowDataTable

Description

Sets or returns whether a data table is displayed in place of the category axis tick mark labels in some chart types.

Signature: VB.NET

```
Property ShowDataTable As Boolean (read/write)
```

Remarks

A data table displays the data from which the chart was generated.

> ⚠ Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a new bar chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsBarChart, _
   0, 2, 2, 10, 10)

'--- First, enable the data table for the chart
MyChart.ShowDataTable = True

'--- The DataTable property is an IChartFrame interface that you can
'--- use to adjust the properties of the chart's data table
'--- Adjust the height and width of the data table
'--- Apply shadowing to the border
MyChart.DataTable.HasShadow = True
MyChart.DataTable.Width = 10
MyChart.DataTable.Height = 10
```

Chart.ShowHorizontal

Description

Sets or returns whether columns are displayed horizontally.

Signature: VB.NET

```
Property ShowHorizontal As Boolean (read/write)
```

Remarks

When set to True, will display columns horizontally, that is, will convert a column chart to a bar chart. ShowHorizontal is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set ShowHorizontal = True to display columns horizontally
MyChart.ShowHorizontal = True
```

Chart.ShowLegend

Description

Sets or returns whether the chart legend is displayed

Signature: VB.NET

```
Property ShowLegend As Boolean (read/write)
```

Remarks

When set to True, displays the chart legend. When set to False, hides the chart legend. ShowLegend is True by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set ShowLegend = False to hide the legend
'--- (legend is visible by default)
MyChart.ShowLegend = True
```

Chart.ShowValueAsPercent

Description

Sets or returns whether the Value Axis values are displayed as percents

Signature: VB.NET

```
Property ShowValueAsPercent As Boolean (read/write)
```

Remarks

When set to True, displays ValueAxis values as percents. ShowValueAsPercent is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'---create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Display value axis values as percents insted of the actual values
MyChart.ShowValuesAsPercent = True
```

Chart.SizeOfCenterHole

Description

Sets or retrieves the size of the center hole of a doughnut chart.

Signature: VB.NET

```
Property SizeOfCenterHole As Integer (read/write)
```

Remarks

By default, SizeOfCenterHole is set to 20% of the doughnut size.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new doughnut chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsDoughnutChart, 0, 2, 2, 10, 20)

'--- For a Doughnut chart, set the percentage size of the center hole
MyChart.SizeOfCenterHole = 50
```

Chart.SliceDistanceFromCenter

Description

Sets or returns the distance of slices from the center of an exploded pie chart

Signature: VB.NET

```
Property SliceDistanceFromCenter As Integer (read/write)
```

Remarks

SliceDistanceFromCenter is set to 20 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new exploded pie chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsPieChart, _
   3, 2, 2, 10, 20)

'--- For an Exploded Pie chart, set the
'--- distance of the slices from center
MyChart.SliceDistanceFromCenter = 100
```

Chart.SpaceBetweenBars

Description

Sets or retrieves the space between data series bars.

Signature: VB.NET

```
Property SpaceBetweenBars As Integer (read/write)
```

Remarks

SpaceBetweenBars is 0 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set the spece between bars of the chart's data series
MyChart.SpaceBetweenBars = 50
```

Chart.SpaceBetweenCategories

Description

Sets or retrieves the space between categories in column and bar charts.

Signature: VB.NET

```
Property SpaceBetweenCategories As Integer (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set the spece between categoties to 50
MyChart.SpaceBetweenCategories = 50
```

Chart.StackDisplayedValues

Description

Sets or returns whether series values will be stacked.

Signature: VB.NET

```
Property StackDisplayValues As Boolean (read/write)
```

Remarks

Stacks series values when set to True.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set this property to True to stack series values
MyChart.StackDisplayValues = True
```

Chart.Subtype

Description

Returns the chart's subtype code.

Signature: VB.NET

```
Property SubType As Integer (read-only)
```

Remarks

For a list of chart subtypes and their codes, see Chart Codes.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and create a new column chart
Dim XlwApp, WrkSht, MyChart, ChartSubType
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Get the numeric value representing the chart's sub-type
'--- Refer to the chart codes to determine the type
ChartSubType = MyChart.SubType
```

Chart.Title

Description

Returns a ChartFrame object representing the chart's title.

Signature: VB.NET

```
Property Title As IChartFrame (read-only)
```

Remarks

The properties of Title are Area, Border, HasShadow, Height, Text, TextFont, TextHorizontalAlignment, TextRotationAngle, TextVerticalAlignment, Width, X, and Y.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Title returns an IChartFrame interface that
'--- can be used to format the chart's title area
'--- Set the chart title and apply a horizontal stripe pattern
MyChart.Title.Text = "My Chart Title"
MyChart.Title.Area.Pattern = 5
```

Chart.Type

Description

Returns the chart's type code.

Signature: VB.NET

```
Property Type As ASXChartType (read-only)
```

Remarks

For a list of chart types and their codes, see Chart Codes.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart, ChartType
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Get a numeric value representing the chart type
ChartType = MyChart.Type
Response.Write "The chart type is: " & ChartType
```

Chart.Valueaxis

Description

Returns an Axis object representing the chart's Y axis.

Signature: VB.NET

```
Property ValueAxis As IAxis (read-only)
```

Remarks

The properties of ValueAxis are AxisLine, CategoryAxisCrossMaxValue, CategoryCrossingValue, LabelFont, LabelPosition, LogarithmicScale, MajorGridline, MajorIncrement, MajorTickType, MaxValue, MinorGridline, MinorIncrement, MinorTickType, MinValue, Number, Title, ValuesInReverseOrder, Wall3D, and Wall3DBorder.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart, ValAxis
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- ValueAxis returns an IAxis interface that represents the value axis
Set ValAxis = MyChart.ValueAxis
```

Chart.View3D

Description

Returns a 3DProperties object representing the properties of a three-dimensional chart.

Signature: VB.NET

```
Property View3D As I3DProperties (read-only)
```

Remarks

The properties of View3D are ChartDepth, DistanceFromEyeToChart, ElevationAngle, PlotVolumeHeight, RotationAngle, SeriesGapDepth, Show3D, Stack3DColumns, UseAutoScaling, UsePerspectiveTransform, and ZAxis.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Get an I3DProperties interface for the chart
'--- to control the 3D characteristics of the chart
'--- Enable 3D by setting Show3D=True
MyChart.View3D.Show3D = True
```

ChartFrame

Description

ChartArea, DataTable, Legend, LabelFrame, PlotArea, and Title are ChartFrame objects.

Signature: VB.NET

```
Public Class ChartFrame
```

Properties

Name	Description

Area	Returns an Area object representing the area of the ChartFrame object.
Border	Returns a Line object representing the border of the ChartFrame object.
DataLabelPlacement	Sets or returns the location of the ChartFrame, if it is a data label, relative to its data point.
HasShadow	Sets or returns whether the ChartFrame object will have a shadow effect.
Height	Sets or returns the vertical offset from the lower right corner row (for ChartArea) or the x-coordinate of the lower left corner of the ChartFrame (for all other ChartFrame objects).
Number	Sets the display format of label numbers for LabelFrame objects.
Text	Sets or returns the text of a Title ChartFrame object.
TextFont	Sets or returns a Font object representing a ChartFrame's font.
TextHorizontalAlignment	Sets or returns the horizontal alignment of text in a ChartFrame.
TextRotationAngle	Sets or returns the rotation angle of text in a ChartFrame.
TextVerticalAlignment	Sets or returns the vertical alignment of text in a ChartFrame.
Width	Sets or returns the horizontal offset from the lower right corner column (for ChartArea objects) or the y-coordinate of the lower left corner of the ChartFrame (for all other ChartFrame objects).
X	Sets or returns the horizontal offset from the upper left corner column (for ChartArea objects) or the x-coordinate of the upper right corner of the ChartFrame (for all other ChartFrame objects).
Y	Sets or returns the vertical offset from the upper left corner row (for ChartArea objects) or the y-coordinate of the upper right corner of the ChartFrame (for all other ChartFrame objects).

ChartFrame.Area

Description

Returns an Area object representing the area of the ChartFrame object.

Signature: VB.NET

```
Property Area As SAArea (read-only)
```

Remarks

The properties of Area are AreaFormatting, BackgroundColor, ForegroundColor, and Pattern.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- IChartFrame interface can be used to format
'--- certain object on a chart
'--- Here, use IChartFrame to format the chart's legend
MyChart.Legend.Area.BackgroundColor = RGB(255, 0, 0)
```

ChartFrame.Border

Description

Returns a Line object representing the border of the ChartFrame object.

Signature: VB.NET

```
Property Border As SALine (read-only)
```

Remarks

The properties of Border are LineColor, LineFormatting, LineStyle, and LineWeight.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
  0, 1, 1, 10, 10)

'--- Format the chart area's border color with the Border property
MyChart.ChartArea.Border.LineColor = RGB(255, 0, 0)
```

ChartFrame.DataLabelPlacement

Description

Sets or returns the location of the ChartFrame, if it is a data label, relative to its data point.

Signature: VB.NET

```
Property DataLabelPlacement As ASXlDataLabelPlacementType (read/write)
```

Remarks

If the specified ChartFrame object is a data label (a label that displays information about a series data point), DataLabelPlacement sets the location of the label relative to its data point.

Set DataLabelPlacement to a ASXlDataLabelPlacementType value. The default value is

saxlDataLabelPlacementDefault (0). In line charts, the default value is equivalent to Right in Microsoft Excel; in pie charts, it is Auto; for bar and column charts, it is Outside; for stacked bar and column charts, it is Center.

ASXlDataLabelPlacementType	Values	
saxlDataLabelPlacementDefault	0	Applies to all chart types.
saxlDataLabelPlacementOutside	1	Applies to bar, column, and pie charts.
saxlDataLabelPlacementInside	2	Applies to bar, column, and pie charts.
saxlDataLabelPlacementCenter	3	Applies to bar, column, line, and pie charts.
saxlDataLabelPlacementAxis	4	Applies to bar and column charts.
saxlDataLabelPlacementAbove	5	Applies to line charts.
saxlDataLabelPlacementBelow	6	Applies to line charts.
saxlDataLabelPlacementLeft	7	Applies to line charts.
saxlDataLabelPlacementRight	8	Applies to line charts.
saxlDataLabelPlacementAuto	9	Applies to pie charts.

Examples

vbscript

```
chrtFrame.DataLabelPlacement = saxlDataLabelPlacementRight
```

ChartFrame.HasShadow

Description

Sets or returns whether the ChartFrame object will have a shadow effect.

Signature: VB.NET

```
Property HasShadow As Boolean (read/write)
```

Remarks

When HasShadow is set to True, the ChartFrame object will have a shadow effect. HasShadow is set to False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
    0, 1, 1, 10, 10)

'--- Show the chart's legend with a shadow effect
MyChart.Legend.HasShadow = True
```

ChartFrame.Height

Description

Sets or returns the vertical offset from the lower right corner row (for ChartArea) or the x-coordinate of the lower left corner of the ChartFrame (for

all other ChartFrame objects).

Signature: VB.NET

```
Property Height As Long (read/write)
```

Remarks

Height has two definitions: one for ChartArea, and another for all other ChartFrame objects.

- For ChartArea: Specifies the ChartArea's vertical offset from its lower right corner row. If the value of Height is negative, the ChartArea's bottom border will be offset upward. If the value of Height is positive, the ChartArea's bottom border will be offset downward.
- For all other ChartFrame objects: Height is the x coordinate of the lower left corner of the ChartFrame, in units of 1/4000 of the ChartArea.

> LegendLocation and the pair of Chartframe properties Height and Width are mutually exclusive. Use either LegendLocation or Height and Width.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Display the ChartArea offset upward of its bottom row
MyChart.ChartArea.Height = -200
```

ChartFrame.Number

Description

Sets the display format of label numbers for LabelFrame objects.

Signature: VB.NET

```
Property Number As Variant (read/write)
```

Remarks

For a complete list of Number codes, see Formatting Codes.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
  0, 1, 1, 10, 10)

'--- Set the format for label numbers to 0.00%
MyChart.ChartArea.Number = 10
```

ChartFrame.Text

Description

Sets or returns the text of a Title ChartFrame object.

Signature: VB.NET

```
Property Text As String (read/write)
```

Remarks

In VBScript, this is the default property for the IChartFrame object.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Set the title string for the chart
MyChart.Title.Text = "Chart Title Here"
```

ChartFrame.TextFont

Description

Sets or returns a Font object representing a ChartFrame's font.

Signature: VB.NET

```
Property TextFont As SAFont (read/write)
```

Remarks

The properties of TextFont are Bold, Charset, Color, Italic, Name, QueryFontDisp, Size, Strikethrough, Subscript, Superscript, and Underline.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, MyTitleFont
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Create a new font
Set MyTitleFont = XlwApp.CreateFont()
MyTitleFont.Name = "Tahoma"
MyTitleFont.Bold = True

'--- Set the title font for the chart
MyChart.Title.TextFont = MyTitleFont
```

ChartFrame.TextHorizontalAlignment

Description

Sets or returns the horizontal alignment of text in a ChartFrame.

Signature: VB.NET

```
Property TextHorizontalAlignment As ASXTextAlignment (read/write)
```

Remarks

Set TextHorizontalAlignment to an ASXTextAlignment value by name or number:

ASXTextAlignment	Values
saxlLeft	1
saxlsCenter	2
saxlRight	3
saxlsJustify	4

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Set the text horizontal alignment
'--- for the chart title to Justify
MyChart.Title.TextHorizontalAlignment = saxlsJustify
```

ChartFrame.TextRotationAngle

Description

Sets or returns the rotation angle of text in a ChartFrame.

Signature: VB.NET

```
Property TextRotationAngle As Long (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Rotate the title text 45 degrees counterclockwise
MyChart.Title.TextRotationAngle = 45
```

ChartFrame.TextVerticalAlignment

Description

Sets or returns the vertical alignment of text in a ChartFrame.

Signature: VB.NET

```
Property TextVerticalAlignment As ASXTextAlignment (read/write)
```

Remarks

The values of TextVerticalAlignment are:

ASXTextAlignment	Number
saxlsTop	1
saxlsCenter	2
saxlsBottom	3
saxlsJustify	4

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Set the text vertical alignment
'--- for the chart title to Bottom
MyChart.Title.TextVerticalAlignment = saxlsBottom
```

ChartFrame.Width

Description

Sets or returns the horizontal offset from the lower right corner column (for ChartArea objects) or the y-coordinate of the lower left corner of the ChartFrame (for all other ChartFrame objects).

Signature: VB.NET

```
Property Width As Long (read/write)
```

Remarks

Width has two definitions: one for ChartArea, and another for all other ChartFrame objects.

- For ChartArea: Specifies the ChartArea's horizontal offset from its lower right corner column. If the value of Width is negative, the ChartArea's right border will be offset to the left. If the value of Width is positive, the ChartArea's right border will be offset to the right. The following example will display the ChartArea offset to the right of its right-border column.
- For all other ChartFrame objects: Width is the y coordinate of the lower left corner of the ChartFrame, in units of 1/4000 of the ChartArea.

> LegendLocation and the pair of ChartFrame properties Height and Width are mutually exclusive. Use either LegendLocation or Height and Width.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Display the ChartArea offset to
'--- the right of its right-border column
MyChart.ChartArea.Width = 200
```

ChartFrame.X

Description

Sets or returns the horizontal offset from the upper left corner column (for ChartArea objects) or the x-coordinate of the upper right corner of the ChartFrame (for all other ChartFrame objects).

Signature: VB.NET

```
Property X As Long (read/write)
```

Remarks

X has two definitions: one for ChartArea, and another for all other ChartFrame objects.

- **For ChartArea:**
  Specifies the ChartArea's horizontal offset from its upper left corner column. If the value of X is negative, the ChartArea's left border will be offset to the left. If the value of X is positive, the ChartArea's left border will be offset to the right. The following example will display the ChartArea offset to the right of its left-border column.

- **For all other ChartFrame objects:**
  X is the x coordinate of the upper right corner of the ChartFrame, in units of 1/4000 of the ChartArea.

> X is not a property of Legend.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
  0, 1, 1, 10, 10)

'--- Display the ChartArea offset to the
'--- right of its left-border column
MyChart.ChartArea.X = 200
```

ChartFrame.Y

Description

Sets or returns the vertical offset from the upper left corner row (for ChartArea objects) or the y-coordinate of the upper right corner of the ChartFrame (for all other ChartFrame objects).

Signature: VB.NET

```
Property Y As Long (read/write)
```

Remarks

Y has two definitions: one for ChartArea, and another for all other ChartFrame objects.

- **For ChartArea:**
  Specifies the ChartArea's vertical offset from its upper left corner row. If the value of Y is negative, the ChartArea's top border will be offset upward. If the value of Y is positive, the ChartArea's top border will be offset downward. The following example will display the ChartArea offset downward of its upper row.

- **For all other ChartFrame objects:**
  Y is the y coordinate of the upper right corner of the ChartFrame, in units of 1/4000 of the ChartArea.

Y is not a property of Legend.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'---Display the ChartArea offset downward of its upper row
MyChart.ChartArea.Y = 200
```

Charts

Description

The collection of charts in a worksheet.

Signature: VB.NET

```
Public Class Charts
```

Examples

vbscript

```
Set Charts = ws.charts
```

Properties

Name	Description
Count	Returns the number of charts in a worksheet.

Indexers

Item	Represents a single chart in a Charts collection.

Methods

Name	Description
Add(ASXChartType, Integer, Long, Long, Long, Long)	The page Chart.Add(ASXChartType, Integer, Long, Long, Long, Long) does not exist.

Charts.Add(ASXChartType, Integer, Long, Long, Long, Long)

Description

Adds a chart to an Excel worksheet.

Signature: VB.NET

```
Function Add(Type As ASXChartType, _
[Subtype As Integer], _
[UpperLeftRow As Long], _
[UpperLeftColumn As Long], _
[LowerRightRow As Long = 10], _
[LowerRightColumn As Long = 10]) As IChart
```

Parameters

Type

Type of chart to add as an ASXChartType

Subtype

(optional) Integer indicating the subtype of chart to add

UpperLeftRow

(optional) Long representing the top row over which to insert the Chart

UpperLeftColumn

(optional) Long representing the leftmost column over which to insert the Chart

LowerRightRow

(optional) Long representing the bottom row over which to insert the Chart

LowerRightColumn

(optional) Long representing the rightmost column over which to insert the Chart

Returns

Returns the Chart that is to be inserted.

Remarks

Consult Chart Codes to select ChartType (pie, column, etc.) and SubType(default 0). UpperLeftRow(default 0), UpperLeftColumn(default 0), LowerRightRow(default 10), and LowerRightColumn(default 10) determine the position of the chart in your spreadsheet. The following example will generate a 3D pie chart with the upper left corner at row 3, column 0 (A), and the lower right corner at row 15, column 7 (G).

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the ICharts object for the first worksheet
Dim XlwApp, ChartsCollection
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set ChartsCollection = XlwApp.Worksheets(1).Charts

'--- Add a new stacked column chart to the worksheet
'--- The chart's upper left corner will be in B2,
'--- the lower right corner will be on J10
ChartsCollection.Add saxlsColumnChart, 0, 2, 2, 10, 10
```

Charts.Count

Description

Returns the number of charts in a worksheet.

Signature: VB.NET

```
Property Count As Integer (read-only)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the SACharts object for the first worksheet
Dim XlwApp, ChartsCollection, ChartCount
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set ChartsCollection = XlwApp.Worksheets(1).Charts

'--- Get a count of the number of charts on the worksheet
ChartCount = ChartsCollection.Count
```

Charts.Item

Description

Represents a single chart in a Charts collection.

Signature: VB.NET

```
Property Item(ChartNum) As IChart
```

Remarks

Item is the default property of Charts, so Charts(1) is equivalent to Charts.Item(1). This collection is 1-based.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the SACharts object for the first worksheet
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Use the default Item property to access a
'--- specific chart in the collection
'--- Get the second chart
Set MyChart = WrkSht.Charts(2)
```

DataLabels

Description

Represents the collection of data labels for a series.

Signature: VB.NET

```
Public Class DataLabels
```

Remarks

A data label displays information about a series data point.

> **Note on Series, LabelFrame, and DataLabels**
> The property values on the LabelFrame object and the label related property values on the Series object will be used as the defaults for all data labels. However, once a data label is accessed through the DataLabels collection, the label no longer be affected by changes to properties on LabelFrame or Series.

Properties

Name	Description
Count	Returns the number of data labels in the DataLabels collection.

Indexers

Item	Returns the ChartFrame at a given position in the DataLabels object.

DataLabels.Count

Description

Returns the number of data labels in the DataLabels collection.

Signature: VB.NET

```
Property Count As Long (read-only)
```

Examples

vbscript

```
DataLabelCount = DataLabels.Count
```

DataLabels.Item

Description

Returns the ChartFrame at a given position in the DataLabels object.

Signature: VB.NET

```
Property Item As ChartFrame (read-only)
```

Remarks

Item is the default property for DataLabels.

Examples

vbscript

```
series.DataLabels(1).Text = "a"
```

DocumentProperties

Description

Represents properties of a specified spreadsheet.

Signature: VB.NET

```
Public Class DocumentProperties
```

Remarks

In Microsoft Excel, these properties are displayed in the Properties sheet. To view the Properties sheet, open Excel's File menu, and select Properties.

The DocumentProperties object is accessible through the ExcelApplication object's DocumentProperties property.

Examples

vbscript

```
Set DocumentProperties = ExcelApplication.DocumentProperties
```

Properties

Name	Description

Application	Returns the Application field from the spreadsheet's Document Properties
Author	Sets or returns a spreadsheet's author.
Category	Sets or returns the value of a spreadsheet's Category field in Microsoft Excel.
Comments	Sets or returns the value of a spreadsheet's Comments field in Microsoft Excel.
Company	Sets or returns the company name displayed in a spreadsheet's Company field in Microsoft Excel.
CreationDate	Sets or returns the creation date of the specified spreadsheet.
Keywords	Sets or returns keywords that can be used to search for the specified spreadsheet.
LastSavedBy	Sets or returns the name of the last person that saved the current spreadsheet.
LastSavedDate	Sets or returns the date on which the current spreadsheet was last saved.
LinksUpToDate	Sets or returns whether the hyperlinks in the spreadsheet are up-to-date.
Manager	Sets or returns the value of a spreadsheet's Manager field in Microsoft Excel.
RevisionNumber	Sets or returns the revision number in the Statistics page of the spreadsheet's Properties sheet.
ScaleCrop	Sets or returns a value that indicates whether scaling or cropping is desired.
Subject	Sets or returns the value of a spreadsheet's Subject field in Microsoft Excel.
Title	Sets or returns the value of a spreadsheet's Title field in Microsoft Excel.

Methods

Name	Description
Clear()	Clears most of the fields in the current spreadsheet's Properties sheet.

DocumentProperties.Application

Description

Returns the Application field from the spreadsheet's Document Properties

Signature: VB.NET

```
Property Application As String (read-only)
```

Remarks

When ExcelWriter saves a generated spreadsheet, Application is set to "SoftArtisans ExcelWriter." Application is a Read Only property.

DocumentProperties.Author

Description

Sets or returns a spreadsheet's author.

Signature: VB.NET

```
Property Author As String (read/write)
```

Remarks

In Microsoft Excel, the value of the Author property is displayed on the Summary page of the spreadsheet's Properties sheet. To see Excel's Author field:

1. In Excel, open the File menu.
2. Select Properties.
3. Select the Summary tab.

Excel allows you to search for a file by author:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Author.
4. In the Value text box, enter the author.
5. Set other search options.
6. Click Search.

Author is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Author = "John Smith"
```

DocumentProperties.Category

Description

Sets or returns the value of a spreadsheet's Category field in Microsoft Excel.

Signature: VB.NET

```
Property Category As String (read/write)
```

Remarks

To see Excel's Category field:

1. In Excel, open the File menu.

2. Select Properties.
3. Select the Summary tab.

You can set Category to group files together. This allows you to search for files according to category:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Category.
4. In the Value text box, enter a category.
5. Set other search options.
6. Click Search.

Category is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Category = "Invoice"
```

DocumentProperties.Clear()

Description

Clears most of the fields in the current spreadsheet's Properties sheet.

Signature: VB.NET

```
Sub Clear()
```

Remarks

To see a file's Properties sheet:

1. In Excel, open the File menu.
2. Select Properties.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Clear
```

DocumentProperties.Comments

Description

Sets or returns the value of a spreadsheet's Comments field in Microsoft Excel.

Signature: VB.NET

```
Property Comments As String (read/write)
```

Remarks

To see Excel Comments field:

1. In Excel, open the File menu.

2. Select Properties.
3. Select the Summary tab.

In Excel, you can use the Comments field to search for files:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Comments.
4. In the Value text box, enter the comments to search for.
5. Set other search options.
6. Click Search.

Comments is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Comments = "Generated by ExcelWriter"
```

DocumentProperties.Company

Description

Sets or returns the company name displayed in a spreadsheet's Company field in Microsoft Excel.

Signature: VB.NET

```
Property Company As String (read/write)
```

Remarks

To see Excel's Company field:

1. In Excel, open the File menu.
2. Select Properties.
3. Select the Summary tab.

Excel allows you to search for files by company:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Company.
4. In the Value text box, enter a company name to search for.
5. Set other search options.
6. Click Search.

Company is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Company = "SoftArtisans"
```

DocumentProperties.CreationDate

Description

Sets or returns the creation date of the specified spreadsheet.

Signature: VB.NET

```
Property CreationDate As Variant (read/write)
```

Remarks

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.

While setting CreationDate will not change the creation date in the file's Properties sheet, you can use the date you set to search for a file by creation date:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select CreationDate.
4. From the Condition list, select a condition.
5. In the Value text box, enter the date to search for.
6. Set other search options.
7. Click Search.

CreationDate is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.CreationDate = "12/20/2003"
```

DocumentProperties.Keywords

Description

Sets or returns keywords that can be used to search for the specified spreadsheet.

Signature: VB.NET

```
Property Keywords As String (read/write)
```

Remarks

In Excel, a file's keywords are displayed in the Summary page of the Properties sheet. To see the Keywords field in Excel:

1. Open Excel's File menu.
2. Select Properties.
3. Select the Summary tab.

To search for files using a keyword:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Keywords.
4. From the Condition list, select a condition.
5. In the Value text box, enter a keyword to search for.
6. Set other search options.

7. Click Search.

Keywords is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Keywords = "ExcelWriter, WordWriter, OfficeWriter"
```

DocumentProperties.LastSavedBy

Description

Sets or returns the name of the last person that saved the current spreadsheet.

Signature: VB.NET

```
Property LastSavedBy As String (read/write)
```

Remarks

In Excel, a file's Last saved by field is displayed in the Statistics page of the file's Properties sheet. To see the field in Excel:

1. Open Excel's File menu.
2. Select Properties.
3. Select the Statistics tab.

You can use the value of LastSavedBy to search for a files:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Last saved by.
4. From the Condition list, select a condition.
5. In the Value text box, enter the date to search for.
6. Set other search options.
7. Click Search.

LastSavedBy is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.LastSavedBy = "Jane"
```

DocumentProperties.LastSavedDate

Description

Sets or returns the date on which the current spreadsheet was last saved.

Signature: VB.NET

```
Property LastSavedDate As Variant (read/write)
```

Remarks

LastSavedDate is not set by default, is not displayed in the spreadsheet's Properties sheet, and cannot be used to search for a file.

LastSavedDate is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.LastSavedDate = "12/20/2003"
```

DocumentProperties.LinksUpToDate

Description

Sets or returns whether the hyperlinks in the spreadsheet are up-to-date.

Signature: VB.NET

```
Property LinksUpToDate As Boolean (read/write)
```

Remarks

The value of LinksUpToDate specifies whether or not the hyperlinks in the spreadsheet are up-to-date. Set LinksUpToDate to true to indicate that the links are up-to-date, and set the property to false to indicate that the links are not up-to-date. LinksUpToDate is set to false by default.

LinksUpToDate is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.LinksUpToDate = True
```

DocumentProperties.Manager

Description

Sets or returns the value of a spreadsheet's Manager field in Microsoft Excel.

Signature: VB.NET

```
Property Manager As String (read/write)
```

Remarks

To see Excel's Manager field:

1. In Excel, open the File menu.
2. Select Properties.
3. Select the Summary tab.

Excel allows you to search for a files by manager:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Manager.
4. In the Value text box, enter the manager name to search for.
5. Set other search options.

6. Click Search.

Manager is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Manager = "John Smith"
```

DocumentProperties.RevisionNumber

Description

Sets or returns the revision number in the Statistics page of the spreadsheet's Properties sheet.

Signature: VB.NET

```
Property RevisionNumber As String (read/write)
```

Remarks

To see Excel's Revision number field:

1. In Excel, open the File menu.
2. Select Properties.
3. Select the Statistics tab.

Excel allows you to search for a files by revision number:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Revision.
4. In the Value text box, enter a revision number.
5. Set other search options.
6. Click Search.

RevisionNumber is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.RevisionNumber = "7"
```

DocumentProperties.ScaleCrop

Description

Sets or returns a value that indicates whether scaling or cropping is desired.

Signature: VB.NET

```
Property ScaleCrop As Boolean (read/write)
```

Remarks

Set ScaleCrop to true when scaling of the thumbnail is desired. Set ScaleCrop to false when cropping is desired.

ScaleCrop is a Read/Write property and is set to true by default.

DocumentProperties.Subject

Description

Sets or returns the value of a spreadsheet's Subject field in Microsoft Excel.

Signature: VB.NET

```
Property Subject As String (read/write)
```

Remarks

To see Excel's Subject field:

1. In Excel, open the File menu.
2. Select Properties.
3. Select the Summary tab.

You can set Subject to group files together. This allows you to search for files according to subject:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Subject.
4. In the Value text box, enter a subject.
5. Set other search options.
6. Click Search.

Subject is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Subject = "Web reporting solutions"
```

DocumentProperties.Title

Description

Sets or returns the value of a spreadsheet's Title field in Microsoft Excel.

Signature: VB.NET

```
Property Title As String (read/write)
```

Remarks

To see Excel's Title field:

1. In Excel, open the File menu.
2. Select Properties.
3. Select the Summary tab.

Excel allows you to search for a files by title:

1. In Excel, click the Search button.

2. Click Advanced Search.
3. From the Property drop-down list, select Title.
4. In the Value text box, enter a title.
5. Set other search options.
6. Click Search.

Title is a Read/Write property.

Examples

vbscript

```
Set DocProperties = ExcelApplication.DocumentProperties
DocProperties.Title = "SoftArtisans OfficeWriter"
```

ExcelApplication

Description

Represents an Excel Workbook in ExcelWriter.

Signature: VB.NET

```
Public Class ExcelApplication
```

Remarks

ExcelApplication is ExcelWriter's principal object. The ExcelApplication object represents an Excel workbook. To generate an Excel spreadsheet, you must create an instance of the ExcelApplication object.

Examples

vbscript

```
Set objExcelApplication = Server.CreateObject("Softartisans.ExcelWriter")
```

Properties

Name	Description
ContentType	Sets or returns the generated spreadsheet's MIME content type.
DecryptPassword	Sets or returns the password used to Decrypt the Workbook.
DocumentProperties	Returns a DocumentProperties object which represents properties of the specified spreadsheet.
EncryptPassword	Sets or returns the password used to encrypt the Workbook.
ShowHScrollBar	Sets or returns whether the spreadsheet will contain a horizontal scroll bar.
ShowVScrollBar	Sets or returns whether the spreadsheet will contain a vertical scroll bar.

SpreadsheetsCreatedToday	Returns the number of spreadsheets ExcelWriter created so far in a 24 hour period.
Style	Returns a built-in style.
Version	Returns the ExcelWriter edition installed.
VersionEx	Returns ExcelWriter's edition and version number.
Worksheets	Returns the collection of Worksheets in the Workbook.

Methods

Name	Description
AnsiToUnicode(String, Long)	The AnsiToUnicode method takes a non-Latin alphabet string and its language's code page and returns a Unicode string.
CreateFont()	Use CreateFont to set a Font object.
CreateStyle()	Use CreateStyle to create a Style object.
NamedRange(String, Long)	Takes a range's Name and returns the Range object.
Open(Variant, Boolean)	Opens an existing spreadsheet.
Save(String, SASaveMethod, SAFileFormat, Response)	The Save method generates a new spreadsheet and saves this spreadsheet to disk, streams it to the client, returns it as an ExcelTemplate object, or returns it in memory as an array of bytes.
UnicodeToAnsi(String, Long)	UnicodeToAnsi takes a Unicode string and the string language's code page and returns an Ansi string.

ExcelApplication.AnsiToUnicode(String, Long)

Description

The AnsiToUnicode method takes a non-Latin alphabet string and its language's code page and returns a Unicode string.

Signature: VB.NET

```
Function AnsiToUnicode(inAnsi As String, CodePage As Long) As String
```

Parameters

inAnsi

Ansi string to be converted to Unicode

CodePage

CodePage of the Ansi string

Returns

A Unicode string representing the converted values of the Ansi input string

Remarks

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.

If the client HTML script's charset property is set to the string's language, and the string is converted to Unicode using AnsiToUnicode, the string will be displayed correctly in the generated spreadsheet.

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

Examples

vbscript

```
'--- The following lines get a Hebrew string from an HTML form,
'--- convert the string to Unicode, and assign the string to cell
'--- E7. The example uses a specific language code page; to
'--- correctly display strings in any language, use code page 65001.
HebrewString = xlw.AnsiToUnicode(Request.Form("FirstName"), 1255)
ws.Cells("E7").Value = HebrewString
```

ExcelApplication.ContentType

Description

Sets or returns the generated spreadsheet's MIME content type.

Signature: VB.NET

```
Property ContentType As String (read/write)
```

Remarks

In most browsers, Microsoft Excel files are mapped to the MIME type "application/vnd.ms-excel" and ExcelWriter's 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.

Examples

vbscript

```
Set xlw = CreateObject("Softartisans.ExcelWriter")
Set ws = xlw.Worksheets(1)
Set cells = ws.Cells
cells.Cell("A1") = "Default content type is '" + xlw.ContentType + "'"
xlw.ContentType = "application/unknown"
cells.Cell("A2") = "New content type is set to '" + xlw.ContentType + "'"
xlw.Save "file.xls", saOpenInPlace
```

ExcelApplication.CreateFont()

Description

Use CreateFont to set a Font object.

Signature: VB.NET

```
Function CreateFont() As SAFont
```

Returns

Returns a new Font object.

Remarks

The Font object represents a font style that you can use when creating a reusable Style, or a single-cell Format. To define a font, call CreateFont, then assign values to the font's properties.

For more information on creating formats and styles, see, Adding Formatting.

Examples

vbscript

```
Set myfont = xlw.CreateFont
myfont.Name = "Arial"
myfont.Size = 10
```

ExcelApplication.CreateStyle()

Description

Use CreateStyle to create a Style object.

Signature: VB.NET

```
Function CreateStyle() As SAStyle
```

Returns

Returns a new Style.

Remarks

The Style object represents a common style for a group of cells. To define a style, first create a Style object, and assign values to any of its properties.

For more information on creating formats and styles, see Adding Formatting to a Spreadsheet.

Examples

vbscript

```
'--- Create a style.
Set mystyle = xlw.CreateStyle
mystyle.Font.Name = "Arial"
mystyle.Font.Bold = True

'--- Assign the style to a cell.
ws.Cells("A1").Style = mystyle
```

ExcelApplication.DecryptPassword

Description

Sets or returns the password used to Decrypt the Workbook.

Signature: VB.NET

```
Property DecryptPassword As String (read/write)
```

Remarks

Before opening an encrypted spreadsheet, set DecryptPassword to the password set using EncryptPassword or in Microsoft Excel (Tools -> Options... -> Security).

Set DecryptPassword before calling Open.

Examples

vbscript

```
Set oXLW = Server.CreateObject("Softartisans.ExcelWriter")
oXLW.DecryptPassword = "MyPassword"
```

ExcelApplication.DocumentProperties

Description

Returns a DocumentProperties object which represents properties of the specified spreadsheet.

Signature: VB.NET

```
Property DocumentProperties SADocumentProperties (read-only)
```

Remarks

In Microsoft Excel, these properties are displayed in the Properties sheet. To view the Properties sheet, open Excel's File menu, and select Properties.

Examples

vbscript

```
Dim XlwApp, DocProps, App, CreateDate

'--- Create an instance of ExcelWriter
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Get a reference to SADocumentProperties
Set DocProps = XlwApp.DocumentProperties

'--- Use DocumentProperties properties
App = DocProps.Application ' returns "SoftArtisans ExcelWriter"
DocProps.Author = "Ms. Jane Doe"
DocProps.Category = "Marketing Category"
DocProps.Comments = "This document contains internal company data."
DocProps.Company = "Product Company, Inc."
CreateDate = DocProps.CreationDate
DocProps.Keywords = "Marketing,Products,Report,Internal"
DocProps.LastSavedBy = "J. Doe"
DocProps.LastSavedDate = "2/24/2004"
DocProps.LinksUpToDate = True
DocProps.Manager = "John Someone"
DocProps.RevisionNumber = "1.1.2"
DocProps.ScaleCrop = False
DocProps.Subject = "Quarterly Marketing Data"
DocProps.Title = "Marketing Report 1Q 2004"

'--- Use the Clear method to clear document properties
DocProps.Clear
```

ExcelApplication.EncryptPassword

Description

Sets or returns the password used to encrypt the Workbook.

Signature: VB.NET

```
Property EncryptPassword As String (read/write)
```

Remarks

If you set EncryptPassword, ExcelWriter will encrypt the generated spreadsheet - when Save is called - using RC4 encryption. 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 Save.

Examples

vbscript

```
Set oXLW = Server.CreateObject("Softartisans.ExcelWriter")
oXLW.EncryptPassword = "MyPassword"
```

ExcelApplication.NamedRange(String, Long)

Description

Takes a range's Name and returns the Range object.

Signature: VB.NET

```
Function NamedRange(Name As String,_
 [WorksheetIndex As Long = -1]) As SARange
```

Parameters

Name

The Name of the Range object.

WorksheetIndex

Index of the worksheet that contains the Range object.

Returns

Returns the specified named range.

Examples

vbscript

```
'--- The following returns the number of Areas within the "Headings" range.
Set Range1 = ws.Cells.Range(1, 1, 1, 4)
Range1.Name = "Headings"
NumHeadingsAreas = xlw.NamedRange("Headings", 1).AreaCount
```

ExcelApplication.Open(Variant, Boolean)

Description

Opens an existing spreadsheet.

Signature: VB.NET

```
Sub Open(FileName As Variant, [aReadOnly As Boolean = False])
```

Parameters

FileName

Can be one of two values: the name of the spreadsheet to open including the path, or a worksheet as an IStream or ByteArray.

aReadOnly

(optional) When set to True, the opened spreadsheet can be read but not modified. That is, you cannot call the Save method later, even if you rename the file.

Remarks

The Open method allows you to use preset spreadsheet formats. For example, you could create an order form spreadsheet that includes headings for item, quantity, price, billing information, etc. Then, to generate an order, you would open the spreadsheet, assign specific values to the appropriate cells, and save the file with a new file name.

Examples

vbscript

```
'--- Open a spreadsheet from a disk location
Dim objExcelApp As SAExcelApplication
Set objExcelApp = CreateObject("SoftArtisans.ExcelWriter")
objExcelApp.Open "c:\XLWFormats\order.xls", True

'--- Assign values to cells
objExcelApp.Save "c:\orders\order00275.xls"
Set objExcelApp = Nothing

'---************************---
'--- Open a spreadsheet from a ByteArray
Set app = Server.CreateObject("SoftArtisans.ExcelWriter")
app.OpenStream byteArray

'---************************---
'--- Open a spreadsheet from an IStream
Set app = Server.CreateObject("Softartisans.ExcelWriter")
app.OpenStream IStream
```

ExcelApplication.Save(String, SASaveMethod, SAFileFormat, Response)

Description

The Save method generates a new spreadsheet and saves this spreadsheet to disk, streams it to the client, returns it as an ExcelTemplate object, or returns it in memory as an array of bytes.

Signature: VB.NET

```
Function Save([FileName],_
 [saveMethod As SASaveMethod],_
 [saveFormat As SAFileFormat = saFileFormatDefault], _
 [Response])
```

Parameters

FileName

(optional) Physical path and file name of the new spreadsheet.

saveMethod

{optional) Specifies whether to save the spreadsheet to disk, open it in Excel, open it in the browser, return an ExcelTemplate object, or return the spreadsheet in memory as an array of bytes.

saveFormat

(optional) Specifies whether to save the spreadsheet in BIFF7 (Excel 95) or BIFF8 (Excel 97/2000) format.

Response

(optional) ASP Response of the web application

Remarks

Assign the SaveMethod parameter by number or name:

0	**saDefault**	If a FileName is assigned, the file will be saved to disk. If a FileName is not assigned, the file will be returned in memory as a sequence of bytes.
1	**saOpenInExcel**	Open the file in Excel.
2	**saOpenInPlace**	Open in the browser.
3	**saOpenAsTemplate**	Return an ExcelTemplate object.

saOpenAsTemplate is not available in ExcelWriterSE.

Assign the FileFormat parameter by name or number:

7	**saFileFormatExcel95**
8	**saFileFormatExcel97** **saFileFormatExcel2000** **saFileFormatExcel2002**

Examples

vbscript

```
'--- Save the spreadsheet with a path and file name
xlw.Save "c:\spreadsheets\workbook1.xls"

'---***************************---
'--- Allow the user to either save the file to disk,
'--- or open it from its current location.
Response.ContentType = "application/vnd.ms-excel"
Response.BinaryWrite(xlw.Save)
```

ExcelApplication.ShowHScrollBar

Description

Sets or returns whether the spreadsheet will contain a horizontal scroll bar.

Signature: VB.NET

```
Property ShowHScrollBar As Boolean (read/write)
```

Remarks

When ShowHScrollBar is set to true (1) (default value), the generated spreadsheet will contain a horizontal scroll bar. Set ShowHScrollBar to false (0) to hide the horizontal scroll bar.

Examples

vbscript

```
xlw.ShowHScrollBar = False
```

ExcelApplication.ShowVScrollBar

Description

Sets or returns whether the spreadsheet will contain a vertical scroll bar.

Signature: VB.NET

```
Property ShowVScrollBar As Boolean (read/write)
```

Remarks

When ShowVScrollBar is set to true (default value), the generated spreadsheet will contain a vertical scroll bar. Set ShowVScrollBar to false to hide the vertical scroll bar.

Examples

vbscript

```
xlw.ShowVScrollBar = False
```

ExcelApplication.SpreadsheetsCreatedToday

Description

Returns the number of spreadsheets ExcelWriter created so far in a 24 hour period.

Signature: VB.NET

```
Property SpreadsheetsCreatedToday As Long (read-only)
```

Remarks

Note that SpreadsheetsCreatedToday contains the number of spreadsheets created by both ExcelApplication and ExcelTemplate.

The hour count begins when the first spreadsheet is created. The count is reset when the first spreadsheet is created after the previous 24 hour period. For example, if you create your first spreadsheet at 10:00 AM on June 2, the 24 hour period will end at 10:00 AM on June 3, and the next period will begin when you create the first spreadsheet after 10:00 AM on June 3.

SpreadsheetsCreatedToday is a Read Only property.

Examples

vbscript

```
ws.Cells("A7").value = "Spreadsheets Created Today: " & _
     xlw.SpreadsheetsCreatedToday
```

ExcelApplication.Style

Description

Returns a built-in style.

Signature: VB.NET

```
Property Style(StyleName As String) As SAStyle (read-only)
```

Parameters

StyleName

String representing the name of the built-in style you wish to return.

Remarks

Use Style to assign one of ExcelWriter's built-in styles to a cell. ExcelWriter includes the following styles:

- Normal
- Comma
- Comma0
- Currency
- Currency0
- Percent
- Time
- Date
- DateTime
- Float
- Hyperlink

Examples

vbscript

```
Set Cells = ws.Cells
Cells.AddHyperlink 7, 1, 2, 2, "http://www.softartisans.com"
Cells(7, 1).Value = "SoftArtisans"
Cells(7, 1).Style = xlw.Style("Hyperlink")
```

ExcelApplication.UnicodeToAnsi(String, Long)

Description

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

Signature: VB.NET

```
Function UnicodeToAnsi(inUnicode As String, CodePage As Long) As String
```

Parameters

inUnicode

Unicode string representing the value to be converted to ANSI

CodePage

CodePage of the output Ansi string

Returns

Unicode String as an Ansi string with the specified code page

Remarks

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.

For a complete list of charset and code page values see Character Set Recognition.

Examples

vbscript

```
<!-- Cell E7 contains a Hebrew string represented in Unicode. The following lines convert the string
from Unicode to Ansi, and display the converted string in the browser. The Hebrew characters will be
displayed correctly. The example uses a specific language code page and charset; to correctly display
strings in any language, use code page 65001 and charset UTF-8.-->

<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)
%>
```

ExcelApplication.Version

Description

Returns the ExcelWriter edition installed.

Signature: VB.NET

```
Property Version As String (read-only)
```

Remarks

Use Version to determine which edition of ExcelWriter is currently installed. For a description of the different editions of ExcelWriter, see Edition Differences. Version returns one of the following values:

saExcelWriter	**0**
saExcelWriterLE	**1**
saExcelWriterSE	**2**

Examples

vbscript

```
Version = XlwApp.Version
```

ExcelApplication.VersionEx

Description

Returns ExcelWriter's edition and version number.

Signature: VB.NET

```
Property VersionEx As String (read-only)
```

Examples

vbscript

```
VersionEx = XlwApp.VersionEx
```

ExcelApplication.Worksheets

Description

Returns the collection of Worksheets in the Workbook.

Signature: VB.NET

```
Property Worksheets As Worksheets (read-only)
```

Remarks

The Worksheets property represents a collection of Worksheet objects. This collection is 1-based.

Examples

vbscript

```
Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set ws = xlw.Worksheets
ws(1).Name = "Data Sheet"
```

Font

Description

Represents a font style that you can use when creating a reusable Style object, or a single-cell Format.

Signature: VB.NET

```
Public Class Font
```

Remarks

To define a font, first set the CreateFont object, then assign values to any or all of its properties.

Examples

vbscript

```
Set myFont = XlwApp.CreateFont()
myFont.Name = "Arial"
myFont.Size = 10
myFont.Color = RGB(0, 0, 255)
```

Properties

Name	Description
Bold	Sets or returns whether text is displayed in bold.
Charset	Sets or returns the character encoding set of a given cell's font.
Color	Sets or returns the font color for the format of a single cell or a reusable Style object.
Italic	Sets or returns whether the text will be displayed in italic
Name	Sets or returns the font of your cell format or Style.
Size	Sets or returns the Font size
Strikethrough	Sets or returns whether the text is displayed with a strikethrough.
Subscript	Sets or returns whether the text is displayed in subscript format
Superscript	Sets or returns whether the text is displayed in superscript format
Underline	Sets or returns an underline style for a specified cell or Style object.

Methods

Name	Description
QueryFontDisp()	Returns the name of the font displayed in a specified cell.

Font.Bold

Description

Sets or returns whether text is displayed in bold.

Signature: VB.NET

```
Property Bold As Boolean (read/write)
```

Remarks

When set to True, displays text in bold.

Examples

vbscript

```
'--- Set Bold to True for the format of a single cell
ws.Cells("A1").Format.Font.Bold = True

'--- Set Bold to True for a reusable style object
Set mystyle = xlw.CreateStyle()
mystyle.Font.Bold = True
```

Font.Charset

Description

Sets or returns the character encoding set of a given cell's font.

Signature: VB.NET

```
Property CharSet As Integer (read/write)
```

Examples

vbscript

```
ws.Cells("B25").Format.Font.Charset = 2  '--- Symbol set
```

Font.Color

Description

Sets or returns the font color for the format of a single cell or a reusable Style object.

Signature: VB.NET

```
Property Color As Long (read/write)
```

Remarks

Defines the font color for the format of a single cell, or for a reusable Style object. Use RGB codes to assign color values.

Examples

vbscript

```
'--- Set Color to RGB 255 (red) for the format of a single cell
ws.Cells("A1").Format.Font.Color = 255

'--- Set Color to RGB 255 (red) for a reusable style object
Set mystyle = xlw.CreateStyle
mystyle.Font.Color = 255
```

Font.Italic

Description

Sets or returns whether the text will be displayed in italic

Signature: VB.NET

```
Property Italic As Boolean (read/write)
```

Remarks

When set to True, displays text in italic.

Examples

vbscript

```
'--- Set Italic to True for the format of a single cell
ws.Cells("A1").Format.Font.Italic = True

'--- Set Italic to True for a reusable style object
Set mystyle = xlw.CreateStyle
mystyle.Font.Italic = True
```

Font.Name

Description

Sets or returns the font of your cell format or Style.

Signature: VB.NET

```
Property Name As String (read/write)
```

Remarks

Use the Name property to assign a font (Arial, Courier, etc.) to your format or style.

Examples

vbscript

```
'--- Set the font to Times New Roman for the format of a single cell
ws.Cells("A1").Format.Font.Name = "Times New Roman"

'--- Set the font to Times New Roman for a reusable style object
Set mystyle = xlw.CreateStyle
mystyle.Font.Name = "Times New Roman"
```

Font.QueryFontDisp()

Description

Returns the name of the font displayed in a specified cell.

Signature: VB.NET

```
Function QueryFontDisp() As Object
```

Returns

Returns the name of the font displayed in a specified cell.

Examples

vbscript

```
'--- Display the font name assigned to the format of a single cell
ws.Cells("B5").Value =  ws.Cells("A5").Format.Font.QueryFontDisp

'--- Display the font name assigned to a reusable Style object
ws.Cells("A14").Value = mystyle.Font.QueryFontDisp
```

Font.Size

Description

Sets or returns the Font size

Signature: VB.NET

```
Property Size As Long (read/write)
```

Remarks

Defines font size.

Examples

vbscript

```
'--- Set font size to 8 pt. for the format of a single cell
ws.Cells("A1").Format.Font.Size = 8

'--- Set font size to 8 pt. for a reusable Style object
Set mystyle = xlw.CreateStyle()
mystyle.Font.Size = 8
```

Font.Strikethrough

Description

Sets or returns whether the text is displayed with a strikethrough.

Signature: VB.NET

```
Property Strikethrough As Boolean (read/write)
```

Remarks

When set to True, displays text in strikethrough format.

Examples

vbscript

```
'--- Set Strikethrough to True for the format of a single cell
ws.Cells("A1").Format.Font.Strikethrough = True

'--- Set Strikethrough to True for a reusable style object
Set mystyle = xlw.CreateStyle
mystyle.Font.Strikethrough = True
```

Font.Subscript

Description

Sets or returns whether the text is displayed in subscript format

Signature: VB.NET

```
Property Subscript As Boolean (read/write)
```

Remarks

When set to True, displays text in subscript format.

Examples

vbscript

```
'--- Set Subscript to True for the format of a single cell
ws.Cells("A1").Format.Font.Subscript = True

'--- Set Subscript to True for a reusable style object
Set mystyle = xlw.CreateStyle
mystyle.Font.Subscript = True
```

Font.Superscript

Description

Sets or returns whether the text is displayed in superscript format

Signature: VB.NET

```
Property Superscript As Boolean (read/write)
```

Remarks

When set to True, displays text in superscript format.

Examples

vbscript

```
'--- Set Superscript to True for the format of a single cell
ws.Cells("A1").Format.Font.Superscript = True

'--- Set Superscript to True for a reusable style object
Set mystyle = xlw.CreateStyle
mystyle.Font.Superscript = True
```

Font.Underline

Description

Sets or returns an underline style for a specified cell or Style object.

Signature: VB.NET

```
Property Underline As SAXlUnderlineStyle (read/write)
```

Remarks

Set Underline to any of the following styles, using either the style's name or number.

Style Name	Style Number	Comments
True	**Preserved for backward compatability.**	Equivalent to saxlUnderline.
False	**Preserved for backward compatability.**	Equivalent to saxlNoUnderline.
saxlUnderline	**-1**	Equivalent to True.
saxlNoUnderline	**0**	Equivalent to False.
saxlSingleUnderline	**1**	For C++. Equivalent to True.
saxlDoubleUnderline	**2**	
saxlAccountingUnderline	**3**	
saxlDoubleAccountingUnderline	**4**	

Examples

vbscript

```
'--- Set Underline to saxlAccountingUnderline for the
'--- format of a single cell
ws.Cells("A1").Format.Font.Underline = saxlAccountingUnderline

'--- Set Underline to saxlDoubleUnderline for a reusable style object
Set mystyle = xlw.CreateStyle
mystyle.Font.Underline = saxlDoubleUnderline
```

Line

Description

Represents the appearance of a chart line.

Signature: VB.NET

```
Public Class Line
```

Remarks

ExcelWriter includes the following line objects:

- Series.Border
- Axis.AxisLine
- Axis.MajorGridline
- Axis.MinorGridline
- Axis.Wall3DBorder
- ChartFrame.Border

Properties

Name	Description
LineColor	Sets or retrieves the color of a Line object (AxisLine, MajorGridline, MinorGridline, Wall3DBorder, Series.Border, or ChartFrame.Border).
LineFormatting	Sets or retrieves the type of formatting applied to an Line object.
LineStyle	Sets or retrieves the style of a Line object (Axis.AxisLine, Axis.MajorGridline, Axis.MinorGridline, Axis.Wall3DBorder, Series.Border, or ChartFrame.Border).
LineWeight	Sets or retrieves the weight of a Line object (Axis.AxisLine, Axis.MajorGridline, Axis.MinorGridline, Axis.Wall3DBorder, Series.Border, or ChartFrame.Border).

Line.LineColor

Description

Sets or retrieves the color of a Line object (AxisLine, MajorGridline, MinorGridline, Wall3DBorder, Series.Border, or ChartFrame.Border).

Signature: VB.NET

```
Property LineColor As Long (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, MySeries
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 1, 1, 10, 10)

'--- Add a series to the chart's series collection
Set MySeries = MyChart.SeriesCollection("A1:B10", , 0)

'--- Use the SALine object to format the series border
MySeries.Border.LineColor = RGB(255, 0, 0)
MySeries.Border.LineStyle = saxlsDash
MySeries.Border.LineFormatting = saxlsCreationCustom
MySeries.Border.LineWeight = saxlsMediumLine

'--- To display a series border and its markers in one solid color, set all of the following
properties to the '--- desired color:
objSeries1.Border.LineColor = rgb(255, 0, 0)
objSeries1.MarkerFormat.BackgroundColor = rgb(255, 0, 0)
objSeries1.MarkerFormat.ForegroundColor = rgb(255, 0, 0)
```

Line.LineFormatting

Description

Sets or retrieves the type of formatting applied to an Line object.

Signature: VB.NET

```
Property LineFormatting As ASXCreationType (read/write)
```

Remarks

The three types of formatting and their codes are:

saxlsCreationAutomatic	0
saxlsCreationNone	1
saxlsCreationCustom	2

When LineFormatting is set to saxlsCreationAutomatic (0), the default formatting settings will be applied to the Line object. SaxlsCreationCustom (2) enables custom formatting assigned to the Line object (colors or style). SaxlsCreationNone will remove all Line formatting (default or custom).

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, MySeries
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 1, 1, 10, 10)

'--- Add a series to the chart's series collection
Set MySeries = MyChart.SeriesCollection("A1:B10", , 0)

'--- Use the SALine object to format the series border
MySeries.Border.LineColor = RGB(255, 0, 0)
MySeries.Border.LineStyle = saxlsDash
MySeries.Border.LineFormatting = saxlsCreationCustom
MySeries.Border.LineWeight = saxlsMediumLine
```

Line.LineStyle

Description

Sets or retrieves the style of a Line object (Axis.AxisLine, Axis.MajorGridline, Axis.MinorGridline, Axis.Wall3DBorder, Series.Border, or ChartFrame.Border).

Signature: VB.NET

```
Property LineStyle As ASXLineStyle (read/write)
```

Remarks

ExcelWriter supports nine line styles:

Solid	0
Dash	1
Dot	2
Dash-Dot	3
Dash-Dot-Dot	4
Dark Gray	5
Medium Gray	6
Light Gray	8

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, MySeries
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 1, 1, 10, 10)

'--- Add a series to the chart's series collection
Set MySeries = MyChart.SeriesCollection("A1:B10", , 0)

'--- Use the SALine object to format the series border
MySeries.Border.LineColor = RGB(255, 0, 0)
MySeries.Border.LineStyle = saxlsDash
MySeries.Border.LineFormatting = saxlsCreationCustom
MySeries.Border.LineWeight = saxlsMediumLine
```

Line.LineWeight

Description

Sets or retrieves the weight of a Line object (Axis.AxisLine, Axis.MajorGridline, Axis.MinorGridline, Axis.Wall3DBorder, Series.Border, or ChartFrame.Border).

Signature: VB.NET

```
Property LineWeight As ASXLineWeight (read/write)
```

Remarks

LineWeight has four possible values:

HairLine	0
SingleLine	1
MediumLine	2
WideLine	3

Examples

vbscript

```
'--- Create an instance of ExcelWriter and add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, MySeries
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, 0, 1, 1, 10, 10)

'--- Add a series to the chart's series collection
Set MySeries = MyChart.SeriesCollection("A1:B10", , 0)

'--- Use the SALine object to format the series border
MySeries.Border.LineColor = RGB(255, 0, 0)
MySeries.Border.LineStyle = saxlsDash
MySeries.Border.LineFormatting = saxlsCreationCustom
MySeries.Border.LineWeight = saxlsMediumLine
```

PageSetup

Description

Represents the page layout properties of a printed worksheet.

Signature: VB.NET

```
Public Class PageSetup
```

Remarks

> 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.

Examples

vbscript

```
Set PageSetup = Worksheet.PageSetup
```

Properties

Name	Description
BlackAndWhite	Sets or returns whether the page will be printing in black and white regardless of whether data are formatting with colors.
BottomMargin	Sets or returns the distance in inches between the worksheet data and the bottom edge of the printed page.
CenterFooter	Sets or returns a center footer that will appear on the printed worksheet.
CenterHeader	Sets or returns a center header that will appear on the printed worksheet.
CenterHorizontally	Sets or returns whether the data are centered horizontally within the margins.
CenterVertically	Sets or returns whether the data are centered vertically within the margins.
Draft	Sets or returns whether Excel will print in draft mode.
FirstPageNumber	Sets or returns the first page number for the printed worksheet.
FitToPagesTall	Sets or returns the number of pages into which the worksheet will be fit.
FitToPagesWide	Sets or returns the number of pages wide the worksheet will be when printed.
FooterMargin	Sets or returns the distance in inches between the footer and the bottom of the page.
HeaderMargin	Sets or returns the distance between the header and the top of the page.

LeftFooter	Sets or returns a left footer that will appear on the printed worksheet.
LeftHeader	Sets or returns a left header that will appear on the printed worksheet.
LeftMargin	Set LeftMargin to specify distance between the worksheet data and the left edge of the printed page.
Order	Sets or returns the order in which the data are numbered and printed when they do not fit on one page.
Orientation	Sets or returns the orientation of the printed worksheet.
PaperSize	Sets or returns the paper size for the printed worksheet.
PrintArea	Sets or returns a worksheet's print area as a Range object.
PrintComments	Sets or returns how notes attached to cells will be printed.
PrintGridlines	Sets or returns whether to print horizontal and vertical cell gridlines on worksheets.
PrintHeadings	Sets or returns whether cell locations are printed in the A1 reference style or the R1C1 reference style
PrintNotes	Sets or returns whether to print notes.
RightFooter	Sets or returns a right footer that will appear on the printed worksheet.
RightHeader	Sets or returns a right header that will appear on the printed worksheet.
RightMargin	Sets or returns the distance in inches between the worksheet data and the right edge of the printed page.
TopMargin	Sets or returns the distance in inches between the worksheet data and the top edge of the printed page.
UseZoom	Returns whether Zoom is enabled on the worksheet.
Zoom	Sets or returns a percentage representing the zoom level of the worksheet.

Methods

Name	Description
GetPrintTitles()	Returns a Range object that contains titles that will be repeated on each page of a printed worksheet.

SetPrintTitleColumns(Long, Long)	Use SetPrintTitleColumns to set title columns that will be repeated on each page of a printed worksheet.
SetPrintTitleRows(Long, Long)	Use SetPrintTitleRows to set title rows that will be repeated on each page of a printed worksheet.

PageSetup.BlackAndWhite

Description

Sets or returns whether the page will be printing in black and white regardless of whether data are formatting with colors.

Signature: VB.NET

```
Property BlackAndWhite As Boolean (read/write)
```

Remarks

Set the BlackAndWhite property to true if you formatted data with colors, but are printing on a black and white printer. If you are using a color printer, setting BlackAndWhite to true may reduce printing time. BlackAndWhite is set to False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Using the worksheet's SAPageSetup object, set
'--- BlackAndWhite to true
WrkSht.PageSetup.BlackAndWhite = True
```

PageSetup.BottomMargin

Description

Sets or returns the distance in inches between the worksheet data and the bottom edge of the printed page.

Signature: VB.NET

```
Property BottomMargin As Double (read/write)
```

Remarks

Set BottomMargin to specify distance in inches between the worksheet data and the bottom edge of the printed page. BottomMargin is set to 0 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the Bottom Margin to 1.25 inches from the worksheet data
WrkSht.PageSetup.BottomMargin = 1.25
```

PageSetup.CenterFooter

Description

Sets or returns a center footer that will appear on the printed worksheet.

Signature: VB.NET

```
Property CenterFooter As String (read/write)
```

Remarks

For instructions on formatting headers and footers see Formatting Headers and Footers.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Using the PageSetup object, set the center footer to "Page X or Y"
WrkSht.PageSetup.CenterFooter = "Page &P of &N"
```

PageSetup.CenterHeader

Description

Sets or returns a center header that will appear on the printed worksheet.

Signature: VB.NET

```
Property CenterHeader As String (read/write)
```

Remarks

For instructions on formatting headers and footers see Formatting Headers and Footers.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Using the PageSetup object, set the center header
WrkSht.PageSetup.CenterHeader = "Your Worksheet's Center Header"
```

PageSetup.CenterHorizontally

Description

Sets or returns whether the data are centered horizontally within the margins.

Signature: VB.NET

```
Property CenterHorizontally As Boolean (read/write)
```

Remarks

When set to True, centers data horizontally within the margins. CenterHorizontally is set to False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set "CenterHorizontally" to "true" to center data within the margins
WrkSht.PageSetup.CenterHorizontally = True
```

PageSetup.CenterVertically

Description

Sets or returns whether the data are centered vertically within the margins.

Signature: VB.NET

```
Property CenterVertically As Boolean (read/write)
```

Remarks

When set to True, centers data vertically within the margins. CenterVertically is set to False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set "CenterVertically" to "true" to center data
'--- vertically within the margins
WrkSht.PageSetup.CenterVertically = True
```

PageSetup.Draft

Description

Sets or returns whether Excel will print in draft mode.

Signature: VB.NET

```
Property Draft As Boolean
```

Remarks

Set Draft to true to reduce printing time. When Draft is set to true, Excel does not print gridlines and most graphics. Draft is set to False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Enable draft printing by setting Draft to True
WrkSht.PageSetup.Draft = True
```

PageSetup.FirstPageNumber

Description

Sets or returns the first page number for the printed worksheet.

Signature: VB.NET

```
Property FirstPageNumber As Integer (read/write)
```

Remarks

By default, the first page will be 1; if the print job does not start with page 1, FirstPageNumber will be the number of the first page in the sequence of pages to print.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Begin page numbering on a number other than 1
WrkSht.PageSetup.FirstPageNumber = 3
```

PageSetup.FitToPagesTall

Description

Sets or returns the number of pages into which the worksheet will be fit.

Signature: VB.NET

```
Property FitToPagesTall As Variant (read/write)
```

Remarks

Adjusts the height of the worksheet or selection from worksheet to fit into a specified number of pages. Use in conjunction with FitToPagesWide. FitToPagesTall is set to 1 by default.

> The properties FitToPagesWide and FitToPagesTall and Zoom cannot be used together.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- When printing, fit all content on two vertical pages
WrkSht.PageSetup.FitToPagesTall = 2
```

PageSetup.FitToPagesWide

Description

Sets or returns the number of pages wide the worksheet will be when printed.

Signature: VB.NET

```
Property FitToPagesWide As Variant (read/write)
```

Remarks

Adjusts the width of the worksheet or selection from worksheet to fit into a specified number of pages. Use in conjunction with FitToPagesTall. FitToPagesWide is set to 1 by default.

The properties FitToPagesWide and FitToPagesTall and Zoom cannot be used together.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- When printing, fit all content on two horizontal pages
WrkSht.PageSetup.FitToPagesWide = 2
```

PageSetup.FooterMargin

Description

Sets or returns the distance in inches between the footer and the bottom of the page.

Signature: VB.NET

```
Property FooterMargin As Double (read/write)
```

Remarks

Use FooterMargin to adjust the distance in inches between the footer and the bottom of the page. The FooterMargin should be smaller than the BottomMargin to prevent the footer from overlapping the data. FooterMargin is set to .5 by default.

Examples

vbscript

```
ws.PageSetup.FooterMargin = .5
```

PageSetup.GetPrintTitles()

Description

Returns a Range object that contains titles that will be repeated on each page of a printed worksheet.

Signature: VB.NET

```
Function GetPrintTitles() As SARange
```

Returns

Returns a Range object that contains titles that will be repeated on each page of a printed worksheet.

Remarks

The range returned by GetPrintTitles includes,

- The area set by SetPrintTitleColumns, if SetPrintTitleRows is not set, OR
- The area set by SetPrintTitleRows, if SetPrintTitleColumns is not set, OR
- A union of the two areas set by SetPrintTitleColumns and SetPrintTitleRows, if both are set

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht, TitleRange
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Get a SARange object that contains the Print Title Rows and Columns
'--- for the worksheet
Set TitleRange = WrkSht.PageSetup.GetPrintTitles()
```

PageSetup.HeaderMargin

Description

Sets or returns the distance between the header and the top of the page.

Signature: VB.NET

```
Property HeaderMargin As Double (read/write)
```

Remarks

Use HeaderMargin to adjust the distance between the header and the top of the page. The HeaderMargin should be smaller than the TopMargin to prevent the header from overlapping the data. HeaderMargin is set to .5 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the worksheet's header margin to 1.5 inches
WrkSht.PageSetup.HeaderMargin = 1.5
```

PageSetup.LeftFooter

Description

Sets or returns a left footer that will appear on the printed worksheet.

Signature: VB.NET

```
Property LeftFooter As String (read/write)
```

Remarks

For instructions on formatting headers and footers see Formatting Headers and Footers.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the left footer for the worksheet
WrkSht.PageSetup.LeftFooter = "The current date is &D"
```

PageSetup.LeftHeader

Description

Sets or returns a left header that will appear on the printed worksheet.

Signature: VB.NET

```
Property LeftHeader As String (read/write)
```

Remarks

For instructions on formatting headers and footers see Formatting Headers and Footers.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the left header for the worksheet
WrkSht.PageSetup.LeftHeader = "The current time is &T"
```

PageSetup.LeftMargin

Description

Set LeftMargin to specify distance between the worksheet data and the left edge of the printed page.

Signature: VB.NET

```
Property LeftMargin As Double (read/write)
```

Remarks

LeftMargin is set to 0 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the left margin to 0.5 inches
WrkSht.PageSetup.LeftMargin = 0.5
```

PageSetup.Order

Description

Sets or returns the order in which the data are numbered and printed when they do not fit on one page.

Signature: VB.NET

```
Property Order As SAXlOrder (read/write)
```

Remarks

Set the Order property to control the order in which data is numbered and printed when they do not fit on one page.

The Order value names and codes are,

saxlDownThenOver	1
saxlOverThenDown	2

The figure below previews the direction that the document will print when you set one of these values. Order is set to saxlDownThenOver by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the page print order to "over then down"
WrkSht.PageSetup.Order = saxlOverThenDown
```

PageSetup.Orientation

Description

Sets or returns the orientation of the printed worksheet.

Signature: VB.NET

```
Property Orientation As SAXlPageOrientation (read/write)
```

Remarks

Assign an Orientation value by its name or code. The Orientation value names and codes are,

saxlPortrait	1
saxlLandscape	2

Orientation is set to saxlPortrait by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the page orientation to Landscape
WrkSht.PageSetup.Orientation = saxlLandscape
```

PageSetup.PaperSize

Description

Sets or returns the paper size for the printed worksheet.

Signature: VB.NET

```
Property PaperSize As SAXlPaperSize (read/write)
```

Remarks

Assign PaperSize by name or number according to the following table.

Name	Number
saxlPaperLetter	**1**
saxlPaperLetterSmall	**2**
saxlPaperTabloid	**3**
saxlPaperLedger	**4**
saxlPaperLegal	**5**
saxlPaperStatement	**6**
saxlPaperExecutive	**7**
saxlPaperA3	**8**
saxlPaperA4	**9**
saxlPaperA4Small	**10**
saxlPaperA5	**11**
saxlPaperB4	**12**

saxlPaperB5	**13**
saxlPaperFolio	**14**
saxlPaperQuarto	**15**
saxlPaper10x14	**16**
saxlPaper11x17	**17**
saxlPaperNote	**18**
saxlPaperEnvelope9	**19**
saxlPaperEnvelope10	**20**
saxlPaperEnvelope11	**21**
saxlPaperEnvelope12	**22**
saxlPaperEnvelope14	**23**
saxlPaperCsheet	**24**
saxlPaperDsheet	**25**
saxlPaperEsheet	**26**
saxlPaperEnvelopeDL	**27**
saxlPaperEnvelopeC5	**28**
saxlPaperEnvelopeC3	**29**
saxlPaperEnvelopeC4	**30**
saxlPaperEnvelopeC6	**31**
saxlPaperEnvelopeC65	**32**
saxlPaperEnvelopeB4	**33**
saxlPaperEnvelopeB5	**34**
saxlPaperEnvelopeB6	**35**
saxlPaperEnvelopeItaly	**36**
saxlPaperEnvelopeMonarch	**37**
saxlPaperEnvelopePersonal	**38**
saxlPaperFanfoldUS	**39**
saxlPaperFanfoldStdGerman	**40**
saxlPaperFanfoldLegalGerman	**41**
saxlPaperUser	**256**

PaperSize is set to saxlPaperLetter by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the paper size to A4
WrkSht.PageSetup.PaperSize = saxlPaperA4
```

PageSetup.PrintArea

Description

Sets or returns a worksheet's print area as a Range object.

Signature: VB.NET

```
Property PrintArea As SARange (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht, PrintRange
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Create the range A1:J20
Set PrintRange = WrkSht.Cells.Range(1,1,20,10)

'--- Assign the range to the PrintArea property
WrkSht.PageSetup.PrintArea = PrintRange
```

PageSetup.PrintComments

Description

Sets or returns how notes attached to cells will be printed.

Signature: VB.NET

```
Property PrintComments As SAXlPrintLocation (read/write)
```

Remarks

Use PrintComments to set your preference on printing the notes that may be attached to cells.

Select saxlPrintSheetEnd to print comments beginning an a separate page at the end of the document. Select PrintInPlace to print comments where they are displayed when you view the worksheet. Please note, if you use PrintInPlace, Excel will only print the comments that are displayed. PrintComments is used when you want precise control of what and where comments are printed. PrintNotes simply prints comments in place or does not print comments. Do not use PrintComments and PrintNotes together.

The PrintComments value names and codes are:

saxlPrintInPlace	**-16**
saxlPrintNoComments	**0**
saxlPrintSheetEnd	**-1**

PrintComments is set to saxlPrintNoComments by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Use the PrintComment property to set the location
'--- of the comments when printing
'--- saxlPrintInPlace prints the comments in place
WrkSht.PageSetup.PrintComments = saxlPrintInPlace
```

PageSetup.PrintGridlines

Description

Sets or returns whether to print horizontal and vertical cell gridlines on worksheets.

Signature: VB.NET

```
Property PrintGridLines As Boolean (read/write)
```

Remarks

Set PrintGridlines to print horizontal and vertical cell gridlines on worksheets. PrintGridlines is set to False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Enable printing of sheet grid lines
WrkSht.PageSetup.PrintGridLines = True
```

PageSetup.PrintHeadings

Description

Sets or returns whether cell locations are printed in the A1 reference style or the R1C1 reference style

Signature: VB.NET

```
Property PrintHeadings As Boolean (read/write)
```

Remarks

Set PrintHeadings to print row numbers and column letters in the A1 reference style or numbered rows and columns in the R1C1 reference style. PrintHeadings is set to False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Enable printing of row and column headings (eg, A1, R1C1)
WrkSht.PageSetup.PrintHeadings = True
```

PageSetup.PrintNotes

Description

Sets or returns whether to print notes.

Signature: VB.NET

```
Property PrintNotes As Boolean (read/write)
```

Remarks

Use PrintNotes to print notes that may be attached to cells. Set PrintNotes to true to print notes where they are displayed when you view the worksheet. (For more precise control of where notes are printed, please see PrintComments.) PrintNotes is set to False by default. Do not use PrintNotes and PrintComments together.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set PrintNotes = True to print any notes associated with cells
WrkSht.PageSetup.PrintNotes = True
```

PageSetup.RightFooter

Description

Sets or returns a right footer that will appear on the printed worksheet.

Signature: VB.NET

```
Property RightFooter As String (read/write)
```

Remarks

For instructions on formatting headers and footers see Formatting Headers and Footers.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the text of the right footer for the worksheet
WrkSht.PageSetup.RightFooter = "This document was printed on: &D"
```

PageSetup.RightHeader

Description

Sets or returns a right header that will appear on the printed worksheet.

Signature: VB.NET

```
Property RightHeader As String (read/write)
```

Remarks

For instructions on formatting headers and footers see Formatting Headers and Footers.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the text of the right header for the worksheet
WrkSht.PageSetup.RightHeader = "This document was printed at time: &T"
```

PageSetup.RightMargin

Description

Sets or returns the distance in inches between the worksheet data and the right edge of the printed page.

Signature: VB.NET

```
Property RightMargin As Double (read/write)
```

Remarks

Set RightMargin to specify distance in inches between the worksheet data and the right edge of the printed page. RightMargin is set to 0 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the right margin to 0.5 inches
WrkSht.PageSetup.RightMargin = 0.5
```

PageSetup.SetPrintTitleColumns(Long, Long)

Description

Use SetPrintTitleColumns to set title columns that will be repeated on each page of a printed worksheet.

Signature: VB.NET

```
Sub SetPrintTitleColumns([FirstColumn As Long = 1], [NumColumns As Long = 1])
```

Parameters

FirstColumn

(optional) The first column for which to set the print title

NumColumns

(optional) The number of columns for which to set the print title

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the print title column at column B
WrkSht.PageSetup.SetPrintTitleColumns 2, 1
```

See also: PageSetup.SetPrintTitleRows(Long, Long) and PageSetup.GetPrintTitles().

PageSetup.SetPrintTitleRows(Long, Long)

Description

Use SetPrintTitleRows to set title rows that will be repeated on each page of a printed worksheet.

Signature: VB.NET

```
Sub SetPrintTitleRows([FirstRow As Long = 1], [NumRow As Long = 1])
```

Parameters

FirstRow

(optional) First row for which to set the print title

NumRow

(optional) Number of rows for which to set the print title

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the print title row at row 3
WrkSht.PageSetup.SetPrintTitleRows 3, 1
```

See also PageSetup.SetPrintTitleColumns(Long, Long) and PageSetup.GetPrintTitles().

PageSetup.TopMargin

Description

Sets or returns the distance in inches between the worksheet data and the top edge of the printed page.

Signature: VB.NET

```
Property TopMargin As Double (read/write)
```

Remarks

Set TopMargin to specify distance in inches between the worksheet data and the top edge of the printed page. TopMargin is set to 0 by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the top margin to 1.5 inches
WrkSht.PageSetup.TopMargin = 1.5
```

PageSetup.UseZoom

Description

Returns whether Zoom is enabled on the worksheet.

Signature: VB.NET

```
Property UseZoom As Boolean (read-only)
```

Remarks

Returns True if Zoom is enabled, and False if Zoom is disabled. If Zoom is disabled, FitToPagesWide and FitToPagesTall are enabled.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Test the UseZoom property to see
'--- what combinatino of FitToPagesWide/Tall and Zoom
'--- should be used
If WrkSht.PageSetup.UseZoom Then
 WrkSht.PageSetup.Zoom = 150
Else
 WrkSht.PageSetup.FitToPagesWide = 2
 WrkSht.PageSetup.FitToPagesTall = 3
End If
```

PageSetup.Zoom

Description

Sets or returns a percentage representing the zoom level of the worksheet.

Signature: VB.NET

```
Property Zoom As Variant (read/write)
```

Remarks

Adjusts the size of the printed worksheet by a specified percentage. The printed worksheet can be enlarged up to 400%, and reduced to 10%. To enable Zoom, set it to True, and assign a percentage value to it. Zoom is set to 100 by default.

> ⚠ Zoom and the pair of properties FitToPagesWide and FitToPagesTall are mutually exclusive.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference
'--- to the first worksheet in the workbook
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set Zoom to 150% of normal size
WrkSht.PageSetup.Zoom = 150
```

See also: UseZoom.

Picture

Description

Represents a single picture in a spreadsheet.

Signature: VB.NET

```
Public Class Picture
```

Remarks

To create a Picture object, use Pictures.CreateFromFile2(Long, Long, String) or Pictures.CreateFromFile(Long, Long, Long, Long, String).

Properties

Name	Description
AreaPattern	Sets or returns the background pattern of a transparent or partly transparent picture.
BorderLineType	Sets or returns the borderline type of a picture.
BorderWeight	Sets or returns the weight of the border line of a picture.
HasShadow	Sets or returns whether Excel displays the picture with a shadow effect.
LowerDeltaX	Sets or returns the picture's horizontal offset from its lower right corner column.
LowerDeltaY	Sets or returns the picture's vertical offset from its lower right corner row.
LowerRightColumn	Returns the lower right column position of an inserted picture.
LowerRightRow	Returns the lower right row position of an inserted picture.
UpperDeltaX	Sets or returns the picture's horizontal offset from its upper left corner column.
UpperDeltaY	Sets or returns the picture's vertical offset from its upper left corner row.
UpperLeftColumn	Returns the upper left column position of an inserted picture.
UpperLeftRow	Returns the upper left row position of an inserted picture.

Methods

Name	Description
Move(Long, Long, Long, Long)	Moves the picture to a specified location.

Picture.AreaPattern

Description

Sets or returns the background pattern of a transparent or partly transparent picture.

Signature: VB.NET

```
Property AreaPattern As ASXAreaPatternType (read/write)
```

Remarks

ExcelWriter supports nineteen patterns:

No Pattern	**0**
0% Pattern	**1**
50% Pattern	**2**
70% Pattern	**3**
25% Pattern	**4**
Dark Horizontal	**5**
Dark Vertical	**6**
Dashed Downward Diagonal	**7**
Dashed Upward Diagonal	**8**
Small Checkerboard	**9**
Trellis	**10**
Light Horizontal	**11**
Light Vertical	**12**
Wide Downward diagonal	**13**
Dark Upward diagonal	**14**
Small Grid	**15**
60% Pattern	**16**
20% Pattern	**17**
10% Pattern	**18**

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- Set the area patter of the picture to Light Vertical
Pic.AreaPattern = saxlsLightVerticalPattern
```

Picture.BorderLineType

Description

Sets or returns the borderline type of a picture.

Signature: VB.NET

```
Property BorderLineType As ASXLineStyle (read/write)
```

Remarks

ExcelWriter supports nine BorderLineTypes:

Solid	**0**
Dash	**1**
Dot	**2**
Dash-Dot	**3**
Dash-Dot-Dot	**4**
Dark Gray	**5**
Medium Gray	**6**
Light Gray	**8**

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- Set the border line type to dash-dot-dot
Pic.BorderLineType = saxlsDash_Dot_Dot
```

Picture.BorderWeight

Description

Sets or returns the weight of the border line of a picture.

Signature: VB.NET

```
Property BorderWeight As ASXLineWeight (read/write)
```

Remarks

BorderWeight has four possible values:

HairLine	**0**
SingleLine	**1**
MediumLine	**2**
WideLine	**3**

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- Set the weight of the picture's border to "medium line"
Pic.BorderWeight = saxlsMediumLine
```

Picture.HasShadow

Description

Sets or returns whether Excel displays the picture with a shadow effect.

Signature: VB.NET

```
Property HasShadow As Boolean (read/write)
```

Remarks

When set to True, displays a picture with a shadow effect. HasShadow is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- Show the picture with a shadow effect (False by default)
Pic.HasShadow = True
```

Picture.LowerDeltaX

Description

Sets or returns the picture's horizontal offset from its lower right corner column.

Signature: VB.NET

```
Property LowerDeltaX As Long (read/write)
```

Remarks

Specifies the picture's horizontal offset from its lower right corner column. If the LowerDeltaX value is negative, the picture's right border will be offset to the left. If the LowerDeltaX value is positive, the picture's right border will be offset to the right.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- The following example will display the picture offset
'--- to the right of its right-border column.
Pic.LowerDeltaX = 200
```

Picture.LowerDeltaY

Description

Sets or returns the picture's vertical offset from its lower right corner row.

Signature: VB.NET

```
Property LowerDeltaY As Long (read/write)
```

Remarks

Specifies the picture's vertical offset from its lower right corner row. If the LowerDeltaY value is negative, the picture's bottom border will be offset upward. If the LowerDeltaY value is positive, the picture's bottom border will be offset downward.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- The following example will display the picture
'--- offset upward of its bottom row
Pic.LowerDeltaY = -200
```

Picture.LowerRightColumn

Description

Returns the lower right column position of an inserted picture.

Signature: VB.NET

```
Property LowerRightColumn As Long (read-only)
```

Remarks

You can use this property in conjunction with LowerDeltaX to correctly position text around an inserted image.

Examples

vbscript

```
Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set ws = xlw.Worksheets(1)
Set pic = ws.Pictures.CreateFromFile2(2, 2, _
 "C:\WINDOWS\WEB\Wallpaper\Boiling Point.jpg")
LowerRightCol = pic.LowerRightColumn
```

Picture.LowerRightRow

Description

Returns the lower right row position of an inserted picture.

Signature: VB.NET

```
Property LowerRightRow As Long (read-only)
```

Remarks

You can use this property in conjunction with LowerDeltaY to correctly position text around an inserted image.

Examples

vbscript

```
Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set ws = xlw.Worksheets(1)
Set pic = ws.Pictures.CreateFromFile2(2, 2, _
 "C:\WINDOWS\WEB\Wallpaper\Boiling Point.jpg")
LowerRightRow = pic.LowerRightRow
```

Picture.Move(Long, Long, Long, Long)

Description

Moves the picture to a specified location.

Signature: VB.NET

```
Sub Move(UpperLeftColumn As Long, _
UpperLeftRow As Long, _
LowerRightColumn As Long, _
LowerRightRow As Long)
```

Parameters

UpperLeftColumn

Leftmost column of the destination of the Picture

UpperLeftRow

Top row of the destination of the Picture

LowerRightColumn

Rightmost column of the destination of the Picture

LowerRightRow

Bottom row of the destination of the Picture

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- Re-position your picture at upper-left corner J9
'--- and lower-right corner V16
Pic.Move 10, 9, 22 16
```

Picture.UpperDeltaX

Description

Sets or returns the picture's horizontal offset from its upper left corner column.

Signature: VB.NET

```
Property UpperDeltaX As Long (read/write)
```

Remarks

Specifies the picture's horizontal offset from its upper left corner column. If the UpperDeltaX value is negative, the picture's left border will be offset to the left. If the UpperDeltaX value is positive, the picture's left border will be offset to the right.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- Display the picture offset to the right
'--- of its left-border column
Pic.UpperDeltaX = 200
```

Picture.UpperDeltaY

Description

Sets or returns the picture's vertical offset from its upper left corner row.

Signature: VB.NET

```
Property UpperDeltaY As Long (read/write)
```

Remarks

Specifies the picture's vertical offset from its upper left corner row. If the UpperDeltaY value is negative, the picture's top border will be offset upward. If the UpperDeltaY value is positive, the picture's top border will be offset downward.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Add a picture to the worksheet at A1
Set Pic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\temp\pic.bmp")

'--- Display the picture offset downward of its upper row
Pic.UpperDeltaY = 200
```

Picture.UpperLeftColumn

Description

Returns the upper left column position of an inserted picture.

Signature: VB.NET

```
Property UpperLeftColumn As Long (read only)
```

Remarks

You can use this property in conjunction with UpperDeltaX to correctly position text around an inserted image.

Examples

vbscript

```
set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
set ws = xlw.Worksheets(1)
Set pic = ws.Pictures.CreateFromFile2(2, 2, _
 "C:\WINDOWS\WEB\Wallpaper\Boiling Point.jpg")
UpperLeftCol = pic.UpperLeftColumn
```

Picture.UpperLeftRow

Description

Returns the upper left row position of an inserted picture.

Signature: VB.NET

```
Property UpperLeftRow As Long (read only)
```

Remarks

You can use this property in conjunction with UpperDeltaY to correctly position text around an inserted image.

Examples

vbscript

```
set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
set ws = xlw.Worksheets(1)
Set pic = ws.Pictures.CreateFromFile2(2, 2, _
 "C:\WINDOWS\WEB\Wallpaper\Boiling Point.jpg")
UpperLeftRow = pic.UpperLeftRow
```

Pictures

Description

The collection of all pictures in a worksheet.

Signature: VB.NET

```
Public Class Pictures
```

Remarks

To create a Pictures object, use Worksheet.Pictures.

Examples

vbscript

```
Set Pictures = WrkSheet.Pictures
```

Properties

Name	Description
Count	Returns the number of pictures in a worksheet.

Indexers

Item	Returns the Picture object at the specified index.

Methods

Name	Description
CreateFromFile(Long, Long, Long, Long, String)	Creates a Picture object from an existing image file.
CreateFromFile2(Long, Long, String)	Creates a Picture object from an existing image file.

Pictures.Count

Description

Returns the number of pictures in a worksheet.

Signature: VB.NET

```
Property Count As Long (read-only)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, PictureCnt
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Get a count of the number of pictures in the worksheet
PictureCount = WrkSht.Pictures.Count
Response.Write "There are " & PictureCnt & " pictures."
```

Pictures.CreateFromFile2(Long, Long, String)

Description

Creates a Picture object from an existing image file.

Signature: VB.NET

```
Function CreateFromFile2(UpperLeftColumn As Long, _
UpperLeftRow As Long, _
FileName As String) As SAPicture
```

Parameters

UpperLeftColumn

Leftmost column into which to insert the image

UpperLeftRow

Top row into which to insert the image

FileName

The complete path and name of the file from which you are creating a picture

Returns

A new Picture object.

Remarks

UpperLeftColumn and UpperLeftRow specify the location in the worksheet of the image's top left corner. FileName specifies the complete path and name of the file from which you are creating a picture.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, PicPath, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

PicPath = "c:\temp\mypic.bmp"

'--- CreateFromFile2 is simliar to CreateFromFile
'--- Except it is only necessary to specify the
'--- upper left corner.  This avoids image distortion
'--- due to unproportional sizing
'--- This picture's upper left corner will be at A1
Set Pic = WrkSht.Pictures.CreateFromFile(1, 1, PicPath)
```

Pictures.CreateFromFile(Long, Long, Long, Long, String)

Description

Creates a Picture object from an existing image file.

Signature: VB.NET

```
Function CreateFromFile(UpperLeftColumn As Long, _
UpperLeftRow As Long, _
LowerRightColumn As Long, _
LowerRightRow As Long, _
FileName As String) As SAPicture
```

Parameters

UpperLeftColumn

Leftmost column into which to insert the picture

UpperLeftRow

Top rowinto which to insert the picture

LowerRightColumn

Rightmost column into which to insert the picture

LowerRightRow

Botom row into which to insert the picture

FileName

Path and name of the file to insert as a picture

Returns

A new Picture object.

Remarks

UpperLeftColumn, UpperLeftRow, LowerRightColumn, and LowerRightRow specify the location of the picture in the worksheet. FileName specifies the complete path and name of the file from which you are creating a picture.

> ⚠ Setting the location in the worksheet of two of the picture's corners may resize the image disproportionately, distorting the image. To avoid image distortion, use Pictures.CreateFromFile2(Long, Long, String).

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, PicPath, Pic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

PicPath = "c:\temp\mypic.bmp"

'--- Use CreateFromFile to add a picture to a worksheet
'--- and specify its dimensions
'--- This picture's upper left corner will be at A1
'--- Its lower right corner at J10
Set Pic = WrkSht.Pictures.CreateFromFile(1, 1, 10, 10, PicPath)
```

Pictures.Item

Description

Returns the Picture object at the specified index.

Signature: VB.NET

```
Property Item(Index As Long) As SAPicture (read-only)
```

Parameters

Index

Numeral index of the Picture to return

Remarks

Represents a single Picture object in a Pictures collection. Item is the default property of Pictures, so Pictures(1) is equivalent to Pictures.Item(1). This collection is 1-based.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and get a reference to worksheet #1
Dim XlwApp, WrkSht, PicOne, PicTwo
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Use the default Item property to get individual SAPictures
'--- in the Pictures collection.  This collection is 1-based.
Set PicOne = WrkSht.Pictures(1)
Set PicTwo = WrkSht.Pictures(2)
```

PivotField

Description

Represents a single field in a PivotTable.

Signature: VB.NET

```
Public Class PivotField
```

Remarks

A pivot field is a category of data that is derived from a field in the PivotTable's source data. To create a PivotField, use the property PivotTable.Fields.

Examples

vbscript

```
'--- The Fields collection is populated automatically
'--- based on the range of data passed to the CreatePivotTable
'--- method.
Set fldProduct = pvtTable.Fields("Product")
```

Properties

Name	Description
AutoShowCount	Sets or returns the number of field items that are displayed in the PivotTable by default.
AutoShowField	Sets or returns ther data field by which top and bottom values are determined.
AutoShowRange	Sets or returns whether the top or bottom values will be shown
AutoShowType	Sets or returns whether AutoShow is enabled.
AutoSortField	Sets or returns the data field by which the field items are sorted.
AutoSortType	Sets or returns whether AutoSorting is enabled.
DataDisplayFormat	Sets the way that values in a data field are displayed.
DisplayName	Sets or returns the name that is displayed on the specified pivot field's label.
InOutlineForm	Sets or returns whether a field is displayed in outline form or tabular form.
IsCalculatedDataField	Returns whether the specified pivot field was added through the AddDataField Method.
Name	Returns the programmatic name of a field
NumberFormat	Sets or returns the display format of number and date fields.
SummarizeBy	Sets or returns the summarization type of the field.
Type	Sets or returns the type of the PivotField.

Methods

Name	Description
AddSubtotal(SAXlSubtotalType)	AddSubtotal inserts a subtotal of the specified field.
ClearSubtotals()	ClearSubtotals clears all subtotal rows/columns inserted through the AddSubtotal method.
RemoveSubtotal(SAXlSubtotalType)	RemoveSubtotal removes a subtotal of the specified field.

PivotField.AddSubtotal(SAXlSubtotalType)

Description

AddSubtotal inserts a subtotal of the specified field.

Signature: VB.NET

```
Function AddSubtotal(newVal As SAXlSubtotalType)
```

Parameters

newVal

Subtotal type representing the type of subtotal to add.

Remarks

You can add several subtotal types. Set the subtotal type to an SAXlSubtotalType value by name or number:

SAXlSubtotalType	Values
saxlSubtotalNone	**0**
saxlSubtotalDefault	**1**
saxlSubtotalSum	**2**
saxlSubtotalCountA	**3**
saxlSubtotalAverage	**4**
saxlSubtotalMax	**5**
saxlSubtotalMin	**6**
saxlSubtotalProduct	**7**
saxlSubtotalCount	**8**
saxlSubtotalStdDev	**9**
saxlSubtotalStdDevp	**10**
saxlSubtotalVar	**11**
saxlSubtotalVarp	**12**

Examples

vbscript

```
fldSales.AddSubtotal saxlSubtotalSum
```

PivotField.AutoShowCount

Description

Sets or returns the number of field items that are displayed in the PivotTable by default.

Signature: VB.NET

```
Property AutoShowCount As Long (read/write)
```

Remarks

AutoShowCount sets the number of field items to display in the PivotTable by default. The property can be set to any value in the range 1-255. AutoShowCount is set to 10 by default.

Either the top or bottom field items will be displayed, according to the value of AutoShowRange. AutoShowField specifies the data field by which top or bottom values are determined.

Examples

vbscript

```
pvtTable.Fields("Month").AutoShowCount = 2
```

PivotField.AutoShowField

Description

Sets or returns ther data field by which top and bottom values are determined.

Signature: VB.NET

```
Property AutoShowField As IPivotField (read/write)
```

Remarks

If AutoShow is enabled (see AutoShowType), only the top or bottom items will be displayed for the specified field in the generated PivotTable. The number of items displayed is determined by the value of AutoShowCount.

AutoShowField specifies the data field by which "top" and "bottom" values are determined. For example, if a PivotTable contains the two data fields "Count of OrderId" (fldOrderId) and "Sum of OrderTotal" (fldOrderTotal), and fldCustomer.AutoShowField is set to fldOrderTotal, the top/bottom fldCustomer.AutoShowCount number of customers will be displayed, by sum of OrderTotal.

AutoShowField can be set to a data field created using either saxlPivotFieldData or AddDataField.

Examples

vbscript

```
fldTotal.Type = saxlPivotFieldData
fldCategory.AutoShowCount = 2
pvtTable.Fields("Country").AutoShowType = saxlAutoShowAutomatic
pvtTable.Fields("Country").AutoShowField = fldTotal
```

PivotField.AutoShowRange

Description

Sets or returns whether the top or bottom values will be shown

Signature: VB.NET

```
Property AutoShowRange As saxlAutoShowRangeType (read/write)
```

Remarks

If AutoShow is enabled (see AutoShowType), only the top or bottom items will be displayed for the specified field in the generated PivotTable. The number of items displayed is determined by the value of AutoShowCount. AutoShowRange specifies whether the top or bottom values will be shown. Set AutoShowRange to an SAXlAutoShowRangeType value by name or number:

SAXlAutoShowRangeType	Values
saxlAutoShowRangeTop	0
saxlAutoShowRangeBottom	1

Examples

vbscript

```
fldCategory.AutoShowRange = saxlAutoShowRangeBottom
```

PivotField.AutoShowType

Description

Sets or returns whether AutoShow is enabled.

Signature: VB.NET

```
Property AutoShowType As SAXlAutoShowType (read/write)
```

Remarks

AutoShowType enables or disables AutoShow. If AutoShow is enabled, only the top or bottom items will be displayed for the specified field in the generated PivotTable. Set AutoShowType to an SAXlAutoShowType value by name or number:

SAXlAutoShowType	Values	
saxlAutoShowManual	0	Turns AutoShow off.
saxlAutoShowAutomatic	1	Turns AutoShow on.

Examples

vbscript

```
fldCategory.AutoShowType = saxlAutoShowAutomatic
```

PivotField.AutoSortField

Description

Sets or returns the data field by which the field items are sorted.

Signature: VB.NET

```
Property AutoSortField As IPivotField (read/write)
```

Remarks

If AutoSorting is enabled, AutoSortField specifies the data field by which to sort the field items.

For example, if:

- A PivotTable contains the two data fields "Count of OrderId" (fldOrderId) and "Sum of OrderTotal" (fldOrderTotal) (in addition to fields that are not data fields)
- fldCustomer.AutoSortType is set to saxlAutoSortDescending
- fldCustomer.AutoSortField is set to fldOrderTotal

Then, the items in the Customer field will be sorted by by sum of OrderTotal, in descending order.

AutoSortField can be set to a data field created using either saxlPivotFieldData or AddDataField.

Examples

vbscript

```
fldCategory.AutoSortType = saxlAutoSortDescending
fldCategory.AutoSortField = fldTotal
```

PivotField.AutoSortType

Description

Sets or returns whether AutoSorting is enabled.

Signature: VB.NET

```
Property AutoSortType As SAXlAutoSortType (read/write)
```

Remarks

AutoSortType enables or disables AutoSorting, and sets the AutoSort order. If AutoSortType is on, the items in the pivot field will be sorted by a specified data field (see AutoSortField) in ascending or descending order.

For example, if:

- A PivotTable contains the two data fields "Count of OrderId" (fldOrderId) and "Sum of OrderTotal" (fldOrderTotal) (in addition to fields that are not data fields)
- fldCustomer.AutoSortType is set to saxlAutoSortDescending
- fldCustomer.AutoSortField is set to fldOrderTotal

Then, the items in the Customer field will be sorted by by sum of OrderTotal, in descending order.

SAXlAutoSortType	Values
saxlAutoSortManual	0
saxlAutoSortAscending	1
saxlAutoSortDescending	2

Examples

vbscript

```
fldCategory.AutoSortType = saxlAutoSortDescending
fldCategory.AutoSortField = fldTotal
```

PivotField.ClearSubtotals()

Description

ClearSubtotals clears all subtotal rows/columns inserted through the AddSubtotal method.

Signature: VB.NET

```
Function ClearSubtotals()
```

Examples

vbscript

```
fldSales.ClearSubtotals()
```

PivotField.DataDisplayFormat

Description

Sets the way that values in a data field are displayed.

Signature: VB.NET

```
Public Property DataDisplayFormat([BaseField As IPivotField], [FieldItem As Long]) As
SAXlDataDisplayFormat(write-only)
```

Parameters

BaseField

(optional) Field on which summaries will be performed.

FieldItem

(optional) Number or SAXlDataDisplayFormat describing the way the values are to be summarized.

Remarks

DataDisplayFormat specifies how to display the values contained in a data field. This property does not affect pivot fields that are not data fields.

A data field contains values to be summarized. By default, number values are summarized by the Sum function and text values are summarized by the Count function. (To change the summarization type set SummarizeBy.) If a PivotTable contains more than one data field, a single field drop-down named Data appears in the report for access to all of the data fields.

Pass DataDisplayFormat an IPivotField and a field item number (both parameters are optional in VBScript). Set DataDisplayFormat to an SAXlDataDisplayFormat value by name or number:

SAXlDataDisplayFormat	Values
saxlDataDisplayNormal	0
saxlDataDisplayDifferenceFrom	1
saxlDataDisplayPercentageOf	2
saxlDataDisplayPercentageDifferenceOf	3
saxlDataDisplayRunningTotalIn	4
saxlDataDisplayPercentageOfRow	5
saxlDataDisplayPercentageOfColumn	6
saxlDataDisplayPercentageOfTotal	7
saxlDataDisplayIndex	8

Examples

vbscript

```
fldSales.DataDisplayFormat(fldSales,6) = saxlDataDisplayPercentageOfTotal
```

PivotField.DisplayName

Description

Sets or returns the name that is displayed on the specified pivot field's label.

Signature: VB.NET

```
Property DisplayName As String (read/write)
```

Remarks

DisplayName sets a name to display on the specified pivot field's label. The default value of this property is is the value of PivotField.Name, unless the field is a data field. The default display name of a data field is "[summarization type] PivotField.Name." For example, if the data in the "Sales" field is summarized by Sum, the display name will be "Sum of Sales."

Examples

vbscript

```
fldProduct.DisplayName = "Products"
```

PivotField.InOutlineForm

Description

Sets or returns whether a field is displayed in outline form or tabular form.

Signature: VB.NET

```
Property InOutlineForm As Boolean (read/write)
```

Remarks

InOutlineForm specifies whether to display a field in outline form or in tabular form.

Examples

vbscript

```
fldProduct.InOutlineForm = true
```

PivotField.IsCalculatedDataField

Description

Returns whether the specified pivot field was added through the AddDataField Method.

Signature: VB.NET

```
Property IsCalculatedDataField As Boolean (read-only)
```

Remarks

IsCalculatedDataField returns true if the specified pivot field is a data field that was added through the AddDataField method.

Examples

vbscript

```
If fldSales.IsCalculatedDataField = true Then
    fldSales.DisplayName = "Sales (Data Field)"
End If
```

PivotField.Name

Description

Returns the programmatic name of a field

Signature: VB.NET

```
Property Name As String (read-only)
```

Remarks

Name returns the programmatic name of a field, the name assigned when the field was created.

PivotField.NumberFormat

Description

Sets or returns the display format of number and date fields.

Signature: VB.NET

```
Property NumberFormat As Variant (read/write)
```

Remarks

NumberFormat sets the display format of number and date fields. For a complete list of format codes, see Formatting Codes.

Examples

vbscript

```
fldTotal.NumberFormat = 8
```

PivotField.RemoveSubtotal(SAXlSubtotalType)

Description

RemoveSubtotal removes a subtotal of the specified field.

Signature: VB.NET

```
Function RemoveSubtotal(newVal As SAXlSubtotalType)
```

Parameters

newVal

Subtotal type of the subtotal you wish to remove.

Remarks

Specify which subtotal to remove by its SAXlSubtotalType:

SAXlSubtotalType	
saxlSubtotalNone	0
saxlSubtotalDefault	1
saxlSubtotalSum	2
saxlSubtotalCountA	3
saxlSubtotalAverage	4
saxlSubtotalMax	5
saxlSubtotalMin	6
saxlSubtotalProduct	7
saxlSubtotalCount	8
saxlSubtotalStdDev	9
saxlSubtotalStdDevp	10
saxlSubtotalVar	11
saxlSubtotalVarp	12

Examples

vbscript

```
fldSales.RemoveSubtotal saxlSubtotalSum
```

PivotField.SummarizeBy

Description

Sets or returns the summarization type of the field.

Signature: VB.NET

```
Property SummarizeBy As SAXlSummarizationType (read/write)
```

Remarks

A data field contains values to be summarized. By default, number values are summarized by the Sum function and text values are summarized by the Count function. To change the summarization type set SummarizeBy to an SAXlSummarizationType by name or number:

SAXlSummarizationType	Values
saxlSummarizationSum	0
saxlSummarizationCount	1
saxlSummarizationAverage	2
saxlSummarizationMax	3
saxlSummarizationMin	4
saxlSummarizationProduct	5

saxlSummarizationCountNums	**6**
saxlSummarizationStdDev	**7**
saxlSummarizationStdDevp	**8**
saxlSummarizationVar	**9**
saxlSummarizationVarp	**10**

Examples

vbscript

```
'--- The Total field will be summarized by sum to show
'--- the total dollar value. The OrderID field will be
'--- counted, to show the number of orders.
fldTotal.SummarizeBy = saxlSummarizationSum
fldOrderID.SummarizeBy = saxlSummarizationCount
```

PivotField.Type

Description

Sets or returns the type of the PivotField.

Signature: VB.NET

```
Property Type As SAXlPivotFieldType (read/write)
```

Remarks

SAXlPivotFieldType	Values
saxlPivotFieldUndefined	**0**
saxlPivotFieldRow	**1**
saxlPivotFieldColumn	**2**
saxlPivotFieldPage	**4**
saxlPivotFieldData	**8**

PivotFields

Description

The collection of fields in a PivotTable.

Signature: VB.NET

```
Public Class PivotFields
```

Remarks

A pivot field is a category of data that is derived from a field in the PivotTable's source data. Use the property PivotTable.Fields to access the PivotFields collection.

Examples

vbscript

```
'--- The Fields collection is populated automatically
'--- based on the range of data passed to the CreatePivotTable
'--- method.
Set fldProduct = pvtTable.Fields("Product")
```

Properties

Name	Description
Count	Returns the number of IPivotFields in the IPivotFields collection.

Indexers

Item	Returns the PivotField at the specified index of the PivotFields collection.

PivotFields.Count

Description

Returns the number of IPivotFields in the IPivotFields collection.

Signature: VB.NET

```
Property Count As Long (read-only)
```

PivotFields.Item

Description

Returns the PivotField at the specified index of the PivotFields collection.

Signature: VB.NET

```
Property Item(Table As Variant) As IPivotField (read-only)
```

Parameters

Table

Number representing the position of the PivotField in the PivotFields collection

Remarks

Item represents a single Pivot field in a IPivotFields collection. In VBScript, Item is the default property of the IPivotFields object, so IPivotFields(1) is equivalent to IPivotFields.Item(1). You can reference an Item either by number or by name.

PivotTable

Description

Represents a single PivotTable in a worksheet.

Signature: VB.NET

```
Public Class PivotTable
```

Remarks

A PivotTable is an interactive table that combines and compares large amounts of data. You can rotate its rows and columns to see different summaries of the source data, and you can display the details of different fields. To create an PivotTable object, use the PivotTables.CreatePivotTable(SARange, Long, Long) method.

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Open the workbook containing source data for the PivotTable.
XlwApp.Open "c:\reports\pivotdata.xls"

'--- Get a reference to the source data sheet.
Set wsData = XlwApp.Worksheets("DataSheet")

'--- Create a second worksheet and name it PivotSheet.
Set wsPivot = XlwApp.Worksheets(2)
wsPivot.Name = "PivotSheet"

'--- The data in the workbook is in a named range called
'--- DataNamedRange.  Get an SARange object representing
'--- that range.
Set rngData = XlwApp.NamedRange("DataNamedRange")

'--- Pass CreatePivotTable the range of source data to
'--- create the PivotTable.
Set pvtTable = wsPivot.PivotTables.CreatePivotTable(rngData, 2, 2)
```

Properties

Name	Description
AutoFormatType	Sets or returns the built-in PivotTable format of the PivotTable.
DataCollectionInRow	Sets or returns whether collection of data fields will be displayed with the row or column fields.
EmptyCellsValue	Sets or returns a string to display in empty cells in the PivotTable.
EnableDrillDown	Sets or returns whether Excel's drilldown capabilities will be active for the PivotTable
EnableFieldDialog	Sets or returns whether the Field dialog for the specified PivotTable will be enabled.
EnableWizard	Sets or returns whether the user will be able to access the Pivot Table wizard from this PivotTable.
Fields	Returns the collection of IPivotFields in the specified PivotTable.
FieldsPerPage	Sets or returns the number of page fields in each row of the PivotTable.

FriendlyErrorValue	Sets or returns the string which replaces error values in the PivotTable.
GrandTotalsForColumns	Sets or returns whether grand totals for each column will be displayed in the PivotTable.
GrandTotalsForRows	Sets or returns whether grand totals for each row will be displayed in the PivotTable.
InOutlineForm	Sets or returns whether the PivotTable is displayed in Outline form, rather than tabular form.
MergeLabels	Sets or returns whether PivotTable lables will be centered vertically and horizontally.
Name	Sets or returns the name of the PivotTable.
PageLayout	Sets or returns the order in which the PivotTable is printed when it does not fit on one page.
PreserveFormatting	Sets or returns whether PivotTable formatting will be preserved when you change the report layout or refresh the data.
RepeatItemLabels	Sets or returns whether item labels will be repeated on each printed page.
SetPrintTitles	Sets or returns whether the PivotTable's row and column labels are displayed as print titles.
ShowFriendlyErrorValue	Sets or returns whether the value by FriendlyErrorValue will replace error values in the PivotTable.
ShowValueForEmptyCells	Sets or returns whether the value set by EmptyCellsValue will be displayed in empty cells in the PivotTable.
SubtotalHiddenPageItems	Sets or returns whether hidden page items will be included in totals.
UpperLeftColumn	Sets or returns the PivotTable's upper left column number.
UpperLeftRow	Sets or returns the PivotTable's upper left row number.

Methods

Name	Description
AddDataField(IPivotField, SAXlSummarizationType)	AddDataField adds a new data field to the PivotTable based on an existing field in the table. Alternatively, you can add a data field by setting an IPivotField's Type to saxlPivotFieldData.
Move(Long, Long)	Move moves the PivotTable to a different location in the worksheet.

PivotTable.AddDataField(IPivotField, SAXlSummarizationType)

Description

AddDataField adds a new data field to the PivotTable based on an existing field in the table. Alternatively, you can add a data field by setting an IPivotField's Type to saxlPivotFieldData.

Signature: VB.NET

```
Function AddDataField(BaseField As IPivotField,
   summFunc As SAXlSummarizationType)
   As IPivotField
```

Parameters

BaseField

Field in the PivotTable on which to base the new field

summFunc

Summarization type value representing the type of summarization you wish to use for the new field.

Returns

A new PivotField based on an existing field in the table.

Remarks

A data field contains values to be summarized. By default, number values are summarized by the Sum function and text values are summarized by the Count function. (To change the summarization type set SummarizeBy.) If a PivotTable contains more than one data field, a single field drop-down named Data appears in the report for access to all of the data fields.

Data fields added are not true fields in that they are not a column of the data range so these fields cannot be changed later to another type.

Pass AddDataField takes two parameters: an IPivotField and an SAXlSummarizationType by name or number.

SAXlSummarizationType	Values
saxlPivotFieldUndefined	0
saxlPivotFieldRow	1
saxlPivotFieldColumn	2
saxlPivotFieldPage	4
saxlPivotFieldData	8

Examples

vbscript

```
Set pvtTable = wsPivot.PivotTables.CreatePivotTable(rngData, 2, 2)

Set fldCountry = pvtTable.Fields("Country")
Set fldCategory = pvtTable.Fields("CategoryName")
Set fldOrderID = pvtTable.Fields("OrderID")
Set fldTotal = pvtTable.Fields("Total")
Set fldYear = pvtTable.Fields("Year")

fldCountry.Type = saxlPivotFieldRow
fldCategory.Type = saxlPivotFieldRow

fldYear.Type = saxlPivotFieldColumn

'--- Add two data fields to the PivotTable.
'--- The Total field will be tabulated by sum to
'--- show the total dollar value. The OrderID
'--- field will be counted, to show the number
'--- of orders.
pvtTable.AddDataField fldTotal, saxlSummarizationSum
pvtTable.AddDataField fldOrderID, saxlSummarizationCount
```

PivotTable.AutoFormatType

Description

Sets or returns the built-in PivotTable format of the PivotTable.

Signature: VB.NET

```
Property AutoFormatType As SAXlAutoFormatType (read/write)
```

Remarks

AutoFormatType applies one of Excel's built-in PivotTable formats to the specified PivotTable. Set AutoFormatType to an SAXlAutoFormatType value by name or number:

SAXlAutoFormatType	Values
saxlAutoFormatNone	-1
saxlAutoFormatClassic1	0
saxlAutoFormatClassic2	1
saxlAutoFormatTable1	2
saxlAutoFormatTable2	3
saxlAutoFormatTable3	4
saxlAutoFormatTable4	5
saxlAutoFormatTable5	6
saxlAutoFormatTable6	7
saxlAutoFormatTable7	8
saxlAutoFormatTable8	9
saxlAutoFormatTable9	10
saxlAutoFormatTable10	11
saxlAutoFormatTable11	12

saxlAutoFormatReport1	**13**
saxlAutoFormatReport2	**14**
saxlAutoFormatReport3	**15**
saxlAutoFormatReport4	**17**
saxlAutoFormatReport5	**18**
saxlAutoFormatReport6	**19**
saxlAutoFormatReport7	**20**
saxlAutoFormatReport8	**21**
saxlAutoFormatReport9	**22**
saxlAutoFormatReport10	**23**
saxlAutoFormatReport11	**24**
saxlAutoFormatReport12	**25**
saxlAutoFormatReport13	**26**
saxlAutoFormatReport14	**27**
saxlAutoFormatReport15	**28**
saxlAutoFormatReport16	**29**
saxlAutoFormatReport17	**30**
saxlAutoFormatReport18	**31**
saxlAutoFormatReport19	**32**
saxlAutoFormatReport20	**33**
saxlAutoFormatReport21	**34**
saxlAutoFormatReport22	**35**
saxlAutoFormatReport23	**36**
saxlAutoFormatBlank	**37**

To apply an AutoFormat in Excel:

1. Select the PivotTable.
2. Open the Format menu.
3. Select AutoFormat.

Examples

vbscript

```
pvtTable.AutoFormatType = saxlAutoFormatReport1
```

PivotTable.DataCollectionInRow

Description

Sets or returns whether collection of data fields will be displayed with the row or column fields.

Signature: VB.NET

```
Property DataCollectionInRow As Boolean (read/write)
```

Remarks

A data field contains values to be summarized. By default, number values are summarized by the Sum function and text values are summarized by the Count function. (To change the summarization type set SummarizeBy.) If a PivotTable contains more than one data field, a single field drop-down named Data appears in the report for access to all of the data fields.

DataCollectionInRow determines whether the collection of data fields will be displayed with the row or column fields. Set this property if the PivotTable contains more than one data field.
The Data button provides access
to the data fields.

Country	Categor	Data
Germany	Beverages	Sum of Total
		Count of OrderID
	Condiment	Sum of Total
		Count of OrderID
	Confection	Sum of Total

Examples

vbscript

```
pvtTable.DataCollectionInRow = True
```

PivotTable.EmptyCellsValue

Description

Sets or returns a string to display in empty cells in the PivotTable.

Signature: VB.NET

```
Property EmptyCellsValue As String (read/write)
```

Remarks

EmptyCellsValue sets a string to display in empty cells in the PivotTable. The string will only be displayed if ShowValueForEmptyCells is set to true.

Examples

vbscript

```
pvtTable.EmptyCellsValue = "N/A"
```

PivotTable.EnableDrillDown

Description

Sets or returns whether Excel's drilldown capabilities will be active for the PivotTable

Signature: VB.NET

```
Property EnableDrillDown As Boolean (read/write)
```

Remarks

If EnableDrillDown is set to true, the user will be able to:

- Select a cell in the PivotTable and click Show Detail (on Excel's PivotTable toolbar) to display detailed source data for the cell.
- Double-click the cell to display detailed source data for the cell.

If EnableDrillDown is set to false, the Show Detail button will be disabled.

Examples

vbscript

```
pvtTable.EnableDrillDown = True
```

PivotTable.EnableFieldDialog

Description

Sets or returns whether the Field dialog for the specified PivotTable will be enabled.

Signature: VB.NET

```
Property EnableFieldDialog As Boolean (read/write)
```

Remarks

EnableFieldDialog enables or disables Excel's PivotTable Field dialog for the specified PivotTable in the generated spreadsheet. If the property is set to true, the user will be able to right-click within the PivotTable and select Field Settings to open the dialog. If the EnableFieldDialog is set to false Field Settings will be disabled.

Examples

vbscript

```
pvtTable.EnableFieldDialog = True
```

PivotTable.EnableWizard

Description

Sets or returns whether the user will be able to access the Pivot Table wizard from this PivotTable.

Signature: VB.NET

```
Property EnableWizard As Boolean (read/write)
```

Remarks

EnableWizard enables or disables:

- Excel's PivotTable and PivotChart Wizard for the specified PivotTable, and
- Excel's PivotTable Options dialog for the specified PivotTable.

If EnableWizard is set to false, the user will not be able to select the PivotTable and open the wizard, or right-click the table and select Table Options...

Examples

$body

vbscript

```
pvtTable.EnableWizard = True
```

PivotTable.Fields

Description

Returns the collection of IPivotFields in the specified PivotTable.

Signature: VB.NET

```
Public ReadOnly Property Fields As IPivotFields
```

Remarks

An IPivotField represents a PivotTable field.

Examples

vbscript

```
'--- The Fields collection is populated automatically
'--- based on the range of data passed to the CreatePivotTable
'--- method.
Set fldProduct = pvtTable.Fields("Product")

'--- Product name will be displayed across the top.
fldProduct.Type = saxlPivotFieldColumn
```

PivotTable.FieldsPerPage

Description

Sets or returns the number of page fields in each row of the PivotTable.

Signature: VB.NET

```
Property FieldsPerPage As Long (read/write)
```

Remarks

If PageLayout is set to saxlOverThenDown, FieldPerPage sets the number of page fields in each row in the PivotTable. If PageLayout is set to saxlDownThenOver, FieldPerPage sets the number of page fields in each column in the PivotTable.

Examples

vbscript

```
pvtTable.FieldsPerPage = 7
```

PivotTable.FriendlyErrorValue

Description

Sets or returns the string which replaces error values in the PivotTable.

Signature: VB.NET

```
Property FriendlyErrorValue As String (read/write)
```

Remarks

FriendlyErrorValue replaces error values in the PivotTable with the specified string. The friendly error string will be displayed only if ShowFriendlyErrorValue is set to true.

Examples

vbscript

```
pvtTable.FriendlyErrorValue = "N/A"
```

PivotTable.GrandTotalsForColumns

Description

Sets or returns whether grand totals for each column will be displayed in the PivotTable.

Signature: VB.NET

```
Property GrandTotalsForColumns As Boolean (read/write)
```

Remarks

If GrandTotalsForColumns is set to true, grand totals for each of the columns will be displayed in the PivotTable. If the property is set to false, grand totals for each of the columns will not be displayed.

Examples

vbscript

```
pvtTable.GrandTotalsForColumns = True
```

PivotTable.GrandTotalsForRows

Description

Sets or returns whether grand totals for each row will be displayed in the PivotTable.

Signature: VB.NET

```
Property GrandTotalsForRows As Boolean (read/write)
```

Remarks

If GrandTotalsForRows is set to true, grand totals for each of the rows will be displayed in the PivotTable. If the property is set to false, grand totals for each of the rows will not be displayed.

Examples

vbscript

```
pvtTable.GrandTotalsForRows = True
```

PivotTable.InOutlineForm

Description

Sets or returns whether the PivotTable is displayed in Outline form, rather than tabular form.

Signature: VB.NET

```
Property InOutlineForm As Boolean (read/write)
```

Remarks

InOutlineForm specifies whether to display the PivotTable in outline form or in tabular form.

Examples

vbscript

```
pvtTable.InOutlineForm = true
```

PivotTable.MergeLabels

Description

Sets or returns whether PivotTable lables will be centered vertically and horizontally.

Signature: VB.NET

```
Property MergeLabels As Boolean (read/write)
```

Remarks

Set MergeLabels to true to center PivotTable labels vertically and horizontally.

Examples

vbscript

```
pvtTable.MergeLabels = True
```

PivotTable.Move(Long, Long)

Description

Move moves the PivotTable to a different location in the worksheet.

Signature: VB.NET

```
Public Sub Move(newUpperLeftRow As Long, newUpperLeftColumn As Long)
```

Parameters

newUpperLeftRow

Top row of the destination location of the PivotTable

newUpperLeftColumn

Leftmost column of the destination location of the PivotTable

Remarks

The method takes the new row and column of the upper left corner of the table. You can also move a PivotTable by setting the properties UpperLeftColumn and UpperLeftRow.

Examples

vbscript

```
pvtTable.Move 7, 7
```

PivotTable.Name

Description

Sets or returns the name of the PivotTable.

Signature: VB.NET

```
Property Name As String (read/write)
```

Remarks

Use Name to set a name for the PivotTable. In Excel the name is displayed in the Pivot Table Options dialog (right-click the PivotTable and select Table Options...). Using the property IPivotTables.Item, you can use the value of Name to reference a specific PivotTable in code.

Examples

vbscript

```
pvtTable.Name = "Sales Report 2003"
```

PivotTable.PageLayout

Description

Sets or returns the order in which the PivotTable is printed when it does not fit on one page.

Signature: VB.NET

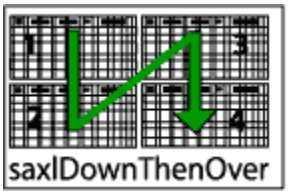
```
Property PageLayout As SAXlOrder (read/write)
```

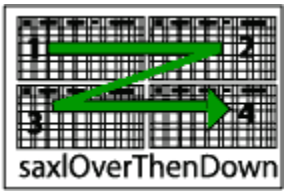
Remarks

Set PageLayout to control the order in which the PivotTable is printed when it does not fit on one page.

Set PageLayout to an SAXlOrder value by name or number:

SAXlOrder	Values
saxlDownThenOver	1
saxlOverThenDown	2





Examples

vbscript

```
pvtTable.PageLayout = saxlOverThenDown
```

PivotTable.PreserveFormatting

Description

Sets or returns whether PivotTable formatting will be preserved when you change the report layout or refresh the data.

Signature: VB.NET

```
Property PreserveFormatting As Boolean (read/write)
```

Remarks

If PreserveFormatting is set to true, most PivotTable formatting will be preserved when you change the report layout or refresh the data. Cell border changes are not preserved.

Examples

vbscript

```
pvtTable.PreserveFormatting = True
```

PivotTable.RepeatItemLabels

Description

Sets or returns whether item labels will be repeated on each printed page.

Signature: VB.NET

```
Property RepeatItemLabels As Boolean (read/write)
```

Remarks

If RepeatItemLabels is set to true, PivotTable item labels will be repeated on each printed page.

Examples

vbscript

```
pvtTable.RepeatItemLabels = True
```

PivotTable.SetPrintTitles

Description

Sets or returns whether the PivotTable's row and column labels are displayed as print titles.

Signature: VB.NET

```
Property SetPrintTitles As Boolean (read/write)
```

Remarks

To display the PivotTable's row and column labels as print titles, set SetPrintTitles to true. Print titles are row or column labels that are displayed at the top or left side of each printed page.

SetPrintTitles will not take effect if Rows to repeat at top or Columns to repeat at left is set. To make sure that these settings are clear:

1. Open the File menu and select Page Setup...
2. Select the Sheet tab.
3. Clear the Rows to repeat at top and Columns to repeat at left fields.

Examples

vbscript

```
pvtTable.SetPrintTitles = True
```

PivotTable.ShowFriendlyErrorValue

Description

Sets or returns whether the value by FriendlyErrorValue will replace error values in the PivotTable.

Signature: VB.NET

```
Property ShowFriendlyErrorValue As Boolean (read/write)
```

Remarks

If ShowFriendlyErrorValue is set to true, the value set by FriendlyErrorValue will replace error values in the PivotTable.

Examples

vbscript

```
pvtTable.ShowFriendlyErrorValue = True
```

PivotTable.ShowValueForEmptyCells

Description

Sets or returns whether the value set by EmptyCellsValue will be displayed in empty cells in the PivotTable.

Signature: VB.NET

```
Property ShowValueForEmptyCells As Boolean (read/write)
```

Remarks

If ShowValueForEmptyCells is set to true, the value set by EmptyCellsValue will be displayed in empty cells in the PivotTable.

Examples

vbscript

```
pvtTable.ShowValueForEmptyCells = True
```

PivotTable.SubtotalHiddenPageItems

Description

Sets or returns whether hidden page items will be included in totals.

Signature: VB.NET

```
Property SubtotalHiddenPageItems As Boolean (read/write)
```

Remarks

If SubtotalHiddenPageItems is set true, hidden page items will be included in totals.

Examples

vbscript

```
pvtTable.SubtotalHiddenPageItems = True
```

PivotTable.UpperLeftColumn

Description

Sets or returns the PivotTable's upper left column number.

Signature: VB.NET

```
Property UpperLeftColumn As Long (read/write)
```

Examples

vbscript

```
pvtTable.UpperLeftColumn = 7
```

PivotTable.UpperLeftRow

Description

Sets or returns the PivotTable's upper left row number.

Signature: VB.NET

```
Property UpperLeftRow As Long (read/write)
```

Examples

vbscript

```
pvtTable.UpperLeftRow = 7
```

PivotTables

Description

The collection of PivotTables in a worksheet.

Signature: VB.NET

```
Public Class PivotTables
```

Remarks

Use the property Worksheet.PivotTables to access the PivotTables collection

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Open the workbook containing source data for the PivotTable.
XlwApp.Open "c:\reports\pivotdata.xls"

'--- Get a reference to the source data sheet.
Set wsData = XlwApp.Worksheets("DataSheet")

'--- Create a second worksheet and name it PivotSheet.
Set wsPivot = XlwApp.Worksheets(2)
wsPivot.Name = "PivotSheet"

'--- The data in the workbook is in a named range called
'--- DataNamedRange.  Get an SARange object representing
'--- that range.
Set rngData = XlwApp.NamedRange("DataNamedRange")

'--- Pass CreatePivotTable the range of source data to
'--- create the PivotTable.
Set pvtTable = wsPivot.PivotTables.CreatePivotTable(rngData, 2, 2)
```

Properties

Name	Description
Count	Returns the number of PivotTables in the IPivotTables collection.

Indexers

Indexers

Item	Returns the PivotTable at the given position in the PivotTables collection

Methods

Name	Description
CreatePivotTable(SARange, Long, Long)	CreatePivotTable takes a range of cells containing source data and creates a new PivotTable.

PivotTables.Count

Description

Returns the number of PivotTables in the IPivotTables collection.

Signature: VB.NET

```
Property Count As Long (read-only)
```

PivotTables.CreatePivotTable(SARange, Long, Long)

Description

CreatePivotTable takes a range of cells containing source data and creates a new PivotTable.

Signature: VB.NET

```
Function CreatePivotTable(aDataRange As SARange,
   aRow As Long,
   aColumn As Long)
   As IPivotTable
```

Parameters

aDataRange

An Range object containing the source data for the PivotTable. The range must be a contiguous region of data with no skipped columns.

aRow

Specifies where to put the PivotTable (the table's upper left corner) in the specified worksheet.
aColumn

aColumn

Specifies where to put the PivotTable (the table's upper left corner) in the specified worksheet.

Returns

A new PivotTable

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Open the workbook containing source data for the PivotTable.
XlwApp.Open "c:\reports\pivotdata.xls"

'--- Get a reference to the source data sheet.
Set wsData = XlwApp.Worksheets("DataSheet")

'--- Create a second worksheet and name it PivotSheet.
Set wsPivot = XlwApp.Worksheets(2)
wsPivot.Name = "PivotSheet"

'--- The data in the workbook is in a named range called
'--- DataNamedRange.  Get an SARange object representing
'--- that range.
Set rngData = XlwApp.NamedRange("DataNamedRange")

'--- Pass CreatePivotTable the range of source data to
'--- create the PivotTable.
Set pvtTable = wsPivot.PivotTables.CreatePivotTable(rngData, 2, 2)
```

PivotTables.Item

Description

Returns the PivotTable at the given position in the PivotTables collection

Signature: VB.NET

```
Property Item(Table As Variant) As IPivotTable (read-only)
```

Parameters

Table

Number representing the position of the PivotTable in the PivotTable collection

Remarks

Item represents a single PivotTable in a IPivotTables collection. In VBScript, Item is the default property of the IPivotTables object, so IPivotTables(1) is equivalent to IPivotTables.Item(1). You can reference an Item either by number or by name.

Range

Description

Represents a range of cells within a worksheet.

Signature: VB.NET

```
Public Class Range
```

Remarks

The Range Object allows you to make a modification to a range of cells within a worksheet, such as the application of a style, by the assignment of methods and properties. To create a Range object, use, Cells.Range, Cells.Rows, or Cells.Columns.

A Range object may include non-adjacent rectangular areas.

Range can be used to assign a certain group of cells to ExcelWriter's PrintArea property. PrintArea is a Range that is read by the PageSetup object when determining the portion of the page that should be printed.

Properties

Name	Description
Area	Returns the Area at the given position in the Range collection.
AreaCount	Returns the number of rectangular areas in a Range object.
Cell	Returns the value of a specified cell in the range.
ColumnCount	Returns the number of columns in the range.
ColumnWidth	Sets the width of the columns in a specified Range.
FirstColumn	Returns the number of the first column in the range.
FirstRow	Returns the number of the first row in the range.
Name	Sets or returns the name of a specified range of cells.
RowCount	Returns the number of rows in the range.
RowHeight	Sets the height of rows in a specified range.

Style	Sets the style properties of the Range of cells using a pre-defined Style object.
UsedRange	Returns a range of used cells within the specified range.
Value	Sets or returns the cell values contained in a Range object as a two-dimensional array.

Indexers

Item	Returns the value of a specified cell in the range.

Methods

Name	Description
AddHyperlink(String, String)	Inserts a hyperlink into a range of cells.
AutoFitHeight()	When WrapText is set to True, AutoFitHeight will adjust the row height of all rows in the range according to the column width.
AutoFitWidth()	Adjusts the width of all columns in the range to the width of their longest values.
BorderAround(SABorderLineStyle, Long)	BorderAround creates a border around the specified range.
GetCellAt(Integer, Integer)	Gets the value contained in a specified cell.
GroupColumns(Long, Long, Boolean)	Outlines specified columns within a range.
GroupRows(Long, Long, Boolean)	Outlines specified rows within a set of cells.
Intersect(SARange)	Creates an intersection of two Range objects.
MergeCells()	Combines a range of specified cells into a single cell.
Union(SARange)	Creates a union of two Range objects.

Range.AddHyperlink(String, String)

Description

Inserts a hyperlink into a range of cells.

Signature: VB.NET

```
Sub AddHyperLink(Address As String[, SubAddress As String])
```

Parameters

Address

Address of the hyperlink

SubAddress

(optional) Sub-address of the hyperlink

Remarks

To insert a hyperlink in a range,

1. Merge the range of cells that will contain the link. This is not required. However, note that all cells in the range will function as a single link. Merging the cells in the range makes it clear that the range is a single unit.
2. Use AddHyperlink to insert a link in the range. AddHyperlink assigns an address and, optionally, a sub-address (e.g. "/search.htm"). The address may be http, https, ftp, mailto, or a UNC path.
3. Assign a value to a cell in the range. The cell value will function as the link.
4. Apply the hyperlink style to the cell containing the link.

Excel 95 does not support this feature.

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

Set Rng = Cells.Range(15, 1, 2, 2)
Rng.MergeCells
Rng.AddHyperlink "http://www.softartisans.com"
Cells(15, 1).Value = "Click here for the SoftArtisans Home Page"
Cells(15, 1).Style = XlwApp.Style("Hyperlink")
```

Range.Area

Description

Returns the Area at the given position in the Range collection.

Signature: VB.NET

```
Property Area(Index As Integer) As SARange (read-only)
```

Parameters

Index

Number representing the position of the Area in the Range collection.

Remarks

A Range may include non-adjacent rectangular areas. The Area property takes an index number, and returns - as a Range object - the corresponding rectangular area within the range.

Examples

vbscript

```
'--- MyRange is an SARange with 2 areas
'--- Get the first area from MyRange
Set rangeOne = MyRange.Area(1)

'--- Get the second area from MyRange
Set rangeTwo = MyRange.Area(2)
```

Range.AreaCount

Description

Returns the number of rectangular areas in a Range object.

Signature: VB.NET

```
Property AreaCount As Integer (read-only)
```

Remarks

A Range may include non-adjacent rectangular areas.

Examples

vbscript

```
'--- MyRange is an SARange object
'--- Get the number of areas in MyRange
NumAreas = MyRange.AreaCount
```

Range.AutoFitHeight()

Description

When WrapText is set to True, AutoFitHeight will adjust the row height of all rows in the range according to the column width.

Signature: VB.NET

```
Sub AutofitHeight()
```

Remarks

To ensure that AutoFitHeight is applied correctly, set the values of the range before setting AutoFitHeight.

Examples

vbscript

```
range.ColumnWidth = 5
Set mystyle = xlw.CreateStyle
mystyle.WrapText = True
range.Style = mystyle
range.AutoFitHeight
```

Range.AutoFitWidth()

Description

Adjusts the width of all columns in the range to the width of their longest values.

Signature: VB.NET

```
Sub AutofitWidth()
```

Remarks

> ⚠ To ensure that AutoFitWidth is applied correctly, set the values of the range before setting AutoFitWidth.

Examples

vbscript

```
range.ColumnWidth = 5
Set mystyle = xlw.CreateStyle
mystyle.WrapText = True
range.Style = mystyle
range.AutoFitWidth
```

Range.BorderAround(SABorderLineStyle, Long)

Description

BorderAround creates a border around the specified range.

Signature: VB.NET

```
Sub BorderAround(BorderLineStyle As SABorderLineStyle, BorderColor As Long)
```

Parameters

BorderLineStyle

LineStyle to assign to the border

BorderColor

Color to assign to the border

Remarks

Set BorderLineStyle to an SABorderLineStyle value by name or number:

SABorderLineStyle	Values
sadgNone	0
sadgThin	1
sadgMedium	2
sadgDashed	3
sadgDotted	4
sadgThick	5
sadgDouble	6
sadgHair	7

sadgMediumDashed	**8**
sadgDashDot	**9**
sadgMediumDashDot	**10**
sadgDashDotDot	**11**
sadgMediumDashDotDot	**12**
sadgSlantedDashDot	**13**

Examples

vbscript

```
'--- This range is A1:B6
Set MyRange = WrkSht.Cells.Range(1, 1, 6, 2)

'--- Apply a thin blue border around cells A1:B6
MyRange.BorderAround sadgThin, RGB(0, 0, 255)
```

Range.Cell

Description

Returns the value of a specified cell in the range.

Signature: VB.NET

```
Property Cell(CellName As String) As SACell (read-only)
```

Parameters

CellName

The name of the cell to be returned, e.g. "A1"

Remarks

Range.cell's reference is relative to the first cell in the range, not the first cell in the worksheet. For example, if the range spans the cells B2: D4, range.Cell("A1") will return the value contained in worksheet.Cells("B2").

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange, FirstCell
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
SAWorksheet WrkSht = XlwApp.Worksheets[1];

'--- This is the range A2:B4
Set MyRange = WrkSht.Cells.Range(2, 1, 2, 2)

'--- Get the first cell in the range
Set FirstCell = MyRange.Cell("A1")
```

Range.ColumnCount

Description

Returns the number of columns in the range.

Signature: VB.NET

```
Property ColumnCount As Integer (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange, ColCount
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
WrkSht = XlwApp.Worksheets(1)

'--- This is the range A2:B4
Set MyRange = WrkSht.Cells.Range(2, 1, 2, 2)

'--- Get the number of columns in the range
ColCount = MyRange.ColumnCount ' returns 2
```

Range.ColumnWidth

Description

Sets the width of the columns in a specified Range.

Signature: VB.NET

```
Property ColumnWidth As Double (write-only)
```

Remarks

Use ColumnWidth to set the width of columns in a specified range.

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
SAWorksheet WrkSht = XlwApp.Worksheets(1)

'--- This range has columns A through D
Set MyRange = WrkSht.Cells.Columns(1, 4)

'--- Set the column width for the range to 10
MyRange.ColumnWidth = 10
```

Range.FirstColumn

Description

Returns the number of the first column in the range.

Signature: VB.NET

```
Property FirstColumn As Integer (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange, FirstCol
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
SAWorksheet WrkSht = XlwApp.Worksheets(1)

'--- This range has columns D through F
Set MyRange = WrkSht.Cells.Columns(4, 3)

'--- Get the first column in the range
FirstCol = MyRange.FirstColumn ' returns 4
```

Range.FirstRow

Description

Returns the number of the first row in the range.

Signature: VB.NET

```
Property FirstRow As Integer (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange, FirstRow
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
SAWorksheet WrkSht = XlwApp.Worksheets(1)

'--- This range has Rows 4 through 6
Set MyRange = WrkSht.Cells.Rows(4, 3)

'--- Get the first column in the range
FirstRow = MyRange.FirstRow ' returns 4
```

Range.GetCellAt(Integer, Integer)

Description

Gets the value contained in a specified cell.

Signature: VB.NET

```
Function GetCellAt(Row As Integer, Column As Integer) As SACell
```

Parameters

Row

Row of the cell to get

Column

Column of the cell to get

Returns

Cell at the given location in the range.

Remarks

> GetCellAt's parameters are relative to the first cell in the range, not the first cell in the worksheet. For example, if the range spans the cells b2:d4, range.GetCellAt(2, 2) will return the value contained in worksheet.Cells("c3").

Examples

vbscript

```
Set FirstRangeCell = range.GetCellAt(1, 1)
```

Range.GroupColumns(Long, Long, Boolean)

Description

Outlines specified columns within a range.

Signature: VB.NET

```
Sub GroupColumns(FirstColumn As Long, _
[NumColumns As Long = 1], _
[Collapsed As Boolean = False])
```

Parameters

FirstColumn

First column to be included in the outline. The first column can be designated as a summary column, (see the Cells.SummaryColumn property) and, if so, should be visible. Columns that follow or precede the summary column contain the outline detail and may be visible or hidden, depending on the value of the collapsed parameter. Column numbering is 1-based.

NumColumns

(optional) Number of columns on which to group

Collapsed

(optional) If set to True, the outline detail columns will be hidden. If set to False, the outline detail columns will be visible.

Remarks

> If you wish to designate the location of the summary column as the last column of the outlined group, then the Cells.SummaryColumn property must be set using the saxlSummaryOnRight (0) value. The summary column should always be visible.

Examples

vbscript

```
'--- Define a range from cell A2 to cell E11
Set range1 = ws.Cells.Range(2, 1, 10, 5)

'--- Outline columns A to D, and hides the outline detail columns.
'--- Column A is the summary column and will be visible. This
'--- assumes that you have set the "SummaryColumn" property to
'--- place summary columns to the left of detail columns. Columns
'--- B to D are the outline detail columns and will be hidden.
range1.GroupColumns 1, 4, True
```

See also: Range.GroupRows(Long, Long, Boolean).

Range.GroupRows(Long, Long, Boolean)

Description

Outlines specified rows within a set of cells.

Signature: VB.NET

```
Sub GroupRows(FirstRow As Long, [NumRows As Long = 1],_
 [Collapsed As Boolean = False])
```

Parameters

FirstRow

First row to be included in the outline. The first row can be designated as a summary row, (see the Cells.SummaryRow property) and, if so, should be visible. Rows that follow or precede the summary row contain the outline detail and may be visible or hidden, depending on the value of the collapsed parameter. Row numbering is 1-based.

NumRows

Number of rows on which to group

Collapsed

If set to True, the outline detail rows will be hidden. If set to False, the outline detail rows will be visible.

Remarks

> If you wish to designate the location of the summary row as the last row of the outlined group, then the Cells.SummaryRow property must be set using the saxlSummaryBelow (0) value. The summary row should always be visible.

Examples

vbscript

```
'--- Define a range from cell A2 to cell E11
Set range1 = ws.Cells.Range(2, 1, 10, 5)

'--- Outline rows 2 to 4, and hide the outline detail rows.
'--- Row 2 is the summary row and will be visible. This assumes
'--- that you have set the "SummaryRow" property to place the
'--- summary row above the detail rows. Rows 3 and 4 are the
'--- outline detail rows and will be hidden.
range1.GroupRows 1, 3, True
```

See also: Range.GroupColumns(Long, Long, Boolean).

Range.Intersect(SARange)

Description

Creates an intersection of two Range objects.

Signature: VB.NET

```
Sub Intersect(Range As SARange)
```

Parameters

Range

Range with which to create the intersection

Examples

vbscript

```
Set range3 = ws1.Cells.Range(22, 7, 5, 4)
Set range4 = ws1.Cells.Range(23, 5, 5, 4)
range3.Intersect range4
```

Range.Item

Description

Returns the value of a specified cell in the range.

Signature: VB.NET

```
Property Item(Row As Variant[, Column As Long]) As SACell (read-only)
```

Parameters

Row

Can either represent the row number, if referencing by coordinates, or the name of the cell by string, e.g. "A1"

Column

The column number, if referencing by coordinates.

Remarks

Item takes either the cell name (e.g. "A1"), or the cell row and column (e.g. (1, 1)). Item is the default property of the Range object, so range(1, 1) is equivalent to range.Item(1, 1).

> ⚠ Item's cell reference is relative to the first cell in the range, not the first cell in the worksheet. For example, if the range spans the cells b2:d4, range.Item("A1") will return the value contained in worksheet.Cells("B2").

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange, FirstCell
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
SAWorksheet WrkSht = XlwApp.Worksheets(1)

'--- This range is B4:E9
Set MyRange = WrkSht.Cells.Range(4, 2, 6, 4)

'--- Get the first cell in the range
Set FirstCell = MyRange(1, 1) ' MyRange("A1") is also valid in VBScript
```

Range.MergeCells()

Description

Combines a range of specified cells into a single cell.

Signature: VB.NET

```
Sub MergeCells()
```

Remarks

Use the address of the upper-left cell in the range to reference the merged cell.

> ⚠ Excel 95 does not support this feature.

Examples

vbscript

```
'--- Create a range and merge the cells into one
Set Rng = Cells.Range(15, 1, 2, 2)
Rng.MergeCells
```

Range.Name

Description

Sets or returns the name of a specified range of cells.

Signature: VB.NET

```
Property Name As String (read/write)
```

Remarks

In an Excel spreadsheet, named ranges are listed in the name box (above the top left corner of the spreadsheet); to move to a named range, select the name from the name list.

A range name must not include spaces.

When assigning a range name, do not use any of the following built-in names:

- Consolidate_Area
- Print_Titles
- Auto_Open
- Recorder
- Auto_Close
- Data_Form
- Extract
- Auto_Activate
- Database
- Auto_Deactivate
- Criteria
- Sheet_Title
- Print_Area

Examples

vbscript

```
WrkSht.Cells("A1").Value = "Last Name"
WrkSht.Cells("B1").Value = "First Name"
WrkSht.Cells("C1").Value = "Address"
WrkSht.Cells("D1").Value = "Phone"

'--- Create a range that contains cells A1:D1
Set HeadingRange = WrkSht.Cells.Range(1, 1, 1, 4)

'--- Create a name for the range
HeadingRange.Name = "Headings"
```

Range.RowCount

Description

Returns the number of rows in the range.

Signature: VB.NET

```
Property RowCount As Integer (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange, NumRows
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
SAWorksheet WrkSht = XlwApp.Worksheets(1)

'--- This range is B4:E9
Set MyRange = WrkSht.Cells.Range(4, 2, 6, 4)

'--- Get the number of rows in the range
NumRows = MyRange.RowCount ' returns 6
```

Range.RowHeight

Description

Sets the height of rows in a specified range.

Signature: VB.NET

```
Property RowHeight As Integer (write-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- This range is rows 2 through 6
Set MyRange = WrkSht.Cells.Rows(2,5)

'--- Set the row height to 10
MyRange.RowHeight = 10
```

Range.Style

Description

Sets the style properties of the Range of cells using a pre-defined Style object.

Signature: VB.NET

```
Property Style As SAStyle (write-only)
```

Remarks

Setting this property allows for the easy application of a pre-defined custom or intrinsic ExcelWriter style to each and every cell that is a member of the range to which the style assignment is made.

If a cell is part of more than one range and each range has a different style, the final applied style will overwrite all previously assigned styles. To overwrite the style for a single cell, use the Cell.Style property. To supplement the previously applied style for a single cell with additional

formatting properties, use the Cell.Format property.

The following example creates a custom "TextStyle" with property settings. It defines a range object (Rng1) that encompasses two rows and spans 5 columns and then assigns the custom "TextStyle" to this range. Finally, it adds an additional formatting property (bold faced font) to Cell "E5" only (which is a member of Rng1).

> It is essential when designing the layout of a workbook to understand the ramifications of using multiple styles and formats and to carefully plan a schema for applying them. For more information on applying styles and formats correctly, see Adding Formatting to a Spreadsheet.

Examples

vbscript

```
Set TextStyle = xlw.CreateStyle
TextStyle.Number = 49
TextStyle.Font = MyFont
TextStyle.WrapText = True
set Rng1 = Ws.Cells.Range(3, 3, 2, 5)
Rng1.Style = TextStyle

Ws.Cells(3,5).Format.Font.Bold = True
```

Range.Union(SARange)

Description

Creates a union of two Range objects.

Signature: VB.NET

```
Sub Union(range As SARange)
```

Parameters

range

Range with which to create a Union.

Examples

vbscript

```
'--- Create a union of two different ranges
Set range4 = ws1.Cells.Range(16, 1, 5, 4)
Set range5 = ws1.Cells.Range(23, 5, 5, 4)
range4.Union range5
```

Range.UsedRange

Description

Returns a range of used cells within the specified range.

Signature: VB.NET

```
Property UsedRange As SARange (read-only)
```

Remarks

If there are no use cells in the range, an empty range is returned.

Examples

vbscript

```
Dim XlwApp, WrkSht, MyRange, UsedRng
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- This range is A1:B6
Set MyRange = WrkSht.Cells.Range(1,1,6,2)

'--- Get the range of used cells
Set UsedRng = MyRange.UsedRange
```

Range.Value

Description

Sets or returns the cell values contained in a Range object as a two-dimensional array.

Signature: VB.NET

```
Property Value As Variant (read/write)
```

Remarks

The following example creates column headings for the first worksheet in a workbook, and uses the Range object's Value property to assign identical headings to the second worksheet.

Examples

vbscript

```
Set ws1 = xlw.Worksheets(1)
Set RngColHeadings1 = ws1.Cells.Range(1, 1, 1, 4)
ws1.Cells("A1").Value = "Last Name"
ws1.Cells("B1").Value = "First Name"
ws1.Cells("C1").Value = "Address"
ws1.Cells("D1").Value = "Phone"

Set ws2 = xlw.Worksheets(2)
Set RngColHeadings2 = ws2.Cells.Range(1, 1, 1, 4)
RngColHeadings2.Value = RngColHeadings1.Value
```

Series

Description

Represents a data series in a chart.

Signature: VB.NET

```
Public Class Series
```

Remarks

To create a Series object, use the SeriesCollection.Add(Area, , Long) method.

Examples

vbscript

```
Set series1 = Chart.SeriesCollection.Add("a1:c1")
```

Properties

Name	Description
Area	Returns an Area object representing the area of a data series.
AxisGroup	Sets or returns the axis type with which the specified series is associated.
Border	Sets or returns a Line object representing the border of a data series.
ChartType	Sets or returns a chart type for the specified series.
ChartSubType	Sets or returns a chart sub-type for the specified series.
DataLabels	Returns a DataLabels object which represents the collection of data labels for the specified series.
ErrorBar	Sets or returns the error bar type for a series.
ErrorBarLine	Returns a Line object representing the appearance of an ErrorBar.
HasSmoothedLine	Sets or returns whether data points in a scatter chart are connected by smoothed lines.
LabelFrame	Returns a ChartFrame object representing the area of the series labels.
LabelShowActualValue	Sets or returns whether the series will include labels displaying actual Y values.
LabelShowCategoryAndValueAsPercent	Sets or returns whether a pie chart series will include labels displaying actual category (X) values and Y values as percents.
LabelShowCategoryLabel	Sets or returns whether a pie chart series will include labels displaying actual category (X) values and Y values as percents.
LabelShowValueAsPercent	Sets or returns whether the series will include labels displaying Y values as percents.
MarkerFormat	Returns an Area object representing the data markers of a line, radar, or scatter series.
MarkerSize	Sets or retrieves the size of series data markers.

MarkerType	Sets or retrieves a data marker type for a series.
Name	Sets or returns the name of a single series.
ScatterValues	Sets or returns the range of cells containing the values of a scatter chart data series.
ShowInLegend	Sets or returns whether the specified series will appear in the chart's legend.
SpaceBetweenBars	Sets or returns the space between the bars of the specified data series.
SpaceBetweenCategories	Sets or returns the space between space between the categories of the specified data series.
Values	Sets or returns the range of cells from which to plot a data series.
XErrorBarCustomMinusValues	Sets or returns a custom minus error value for the X ErrorBar.
XErrorBarCustomPlusValues	Sets or returns a custom plus error value for the X ErrorBar.
XErrorBarTShaped	Sets or returns the shape of the X ErrorBar.
XErrorBarValue	Sets or returns the error value of an X ErrorBar.
XErrorBarValueSource	Sets or returns the type of error value that the X ErrorBar will display.
YErrorBarCustomMinusValues	Sets or returns a custom minus error value for the Y ErrorBar.
YErrorBarCustomPlusValues	Sets or returns a custom plus error value for the Y ErrorBar.
YErrorBarTShaped	Sets or returns the shape of the Y ErrorBar.
YErrorBarValue	Sets or returns the error value of a Y ErrorBar.
YErrorBarValueSource	Sets or returns the type of error value that the Y ErrorBar will display.

Series.Area

Description

Returns an Area object representing the area of a data series.

Signature: VB.NET

```
Property Area As SAArea (read-only)
```

Remarks

The properties of Area are AreaFormatting, BackgroundColor, ForegroundColor, and Pattern.

Examples

vbscript

```
'--- Create an instance of ExcelWriter
'--- and add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- The Area property returns an SAArea object
'--- you can use to format
'--- the data series area
'--- Set the foreground color of a data series area to red.
'--- ColorTranslator and Color are in the System.Drawing namespace
Series.Area.ForegroundColor = RGB(255, 0, 0)
```

Series.AxisGroup

Description

Sets or returns the axis type with which the specified series is associated.

Signature: VB.NET

```
Property AxisGroup As ASXAxisGroup (read/write)
```

Remarks

The AxisGroup property is only available for the chart type Column Line 2 Axes. AxisGroup sets or returns the axis type - primary or secondary - that the specified series is associated with. Set AxisGroup to an ASXAxisGroup value, by name or number:

ASXAxisGroup	Values
saxlsPrimaryAxis	1
saxlsSecondaryAxis	2

Examples

vbscript

```
'--- Create an instance of ExcelWriter
'--- and add a column line 2 axes chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnLine2AxesChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- For ColumnLine2Axes charts, use the AxisGroup property
'--- to determine which axis a series will represent
'--- Set the series "Series" to the second axis
Series.AxisGroup = saxlsSecondaryAxis
```

Series.Border

Description

Sets or returns a Line object representing the border of a data series.

Signature: VB.NET

```
Property Border As SALine (read/write)
```

Remarks

The properties of Border are LineColor, LineFormatting, LineStyle, and LineWeight.

To display a series border and its markers in one solid color, set all of the following properties to the desired color:

- Series.Border.LineColor
- Series.MarkerFormat.BackgroundColor
- Series.MarkerFormat.ForegroundColor

Examples

vbscript

```
'--- Create an instance of ExcelWriter
'--- and add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Use the Border property to format series borders
Series.Border.LineColor = RGB(255, 0, 0)
Series.MarkerFormat.BackgroundColor = RGB(255, 0, 0)
Series.MarkerFormat.ForegroundColor = RGB(255, 0, 0)
```

Series.ChartSubType

Description

Sets or returns a chart sub-type for the specified series.

Signature: VB.NET

```
Property ChartSubType As Integer (read/write)
```

Remarks

The ChartSubType property is only available for the chart types Column Line and Column Line 2 Axes. The sub-type specifies whether the series is not stacked, stacked, or 100% stacked. You will find chart type and sub-type images in Chart Codes.

ChartSubType is a Read/Write value.

ChartSubType	Values
Not stacked	0 (default)
saxlsStacked	1
saxls100PercentStacked	2

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column line chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnLineChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- For Column Line and Column Line 2 Axes charts
'--- Use ChartType to set the chart type fo the series
'--- Set the chart type to Line Chart
Series.ChartType = saxlsLineChart

'--- Also, set the chart subtype
Series.ChartSubType = saxlsStacked
```

Series.ChartType

Description

Sets or returns a chart type for the specified series.

Signature: VB.NET

```
Property ChartType As ASXChartType (read/write)
```

Remarks

The ChartType property is only available for the chart types Column Line and Column Line 2 Axes.
ASXChartType Values for Series.ChartType

saxlsColumnChart	**0**
saxlsLineChart	**2**

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column line chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnLineChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- For Column Line and Column Line 2 Axes charts
'--- Use ChartType to set the chart type fo the series
'--- Set the chart type to Line Chart
Series.ChartType = saxlsLineChart
```

Series.DataLabels

Description

Returns a DataLabels object which represents the collection of data labels for the specified series.

Signature: VB.NET

```
Property DataLabels As IDataLabels (read-only)
```

Remarks

A data label displays information about a series data point.

Examples

vbscript

```
Dim xlw
Dim ws
Dim chart
Dim series

Set xlw = Server.CreateObject("SoftArtisans.ExcelWriter")
Set ws = xlw.Worksheets(1)
Set chart = ws.Charts.Add(7, 3, 3, 5, 20, 18)
Set series = chart.SeriesCollection(1)
series.DataLabels(1).Text = "A"
series.DataLabels(2).Text = "B"
series.DataLabels(3).Text = "C"
```

Series.ErrorBar

Description

Sets or returns the error bar type for a series.

Signature: VB.NET

```
Property ErrorBar As ASXErrorBarType (read/write)
```

Remarks

Error bars display potential error, in relation to data points. The following chart types may include Y error bars: Bar, Two-dimensional Area, Column, Line, and Scatter. X error bars will display only in Scatter charts. ErrorBar has five possible values:

saxlsErrorBarNone	**0 (Default value)**
saxlsXDirectionPlus	**1**
saxlsXDirectionMinus	**2**
saxlsYDirectionPlus	**4**
saxlsYDirectionMinus	**8**

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
    0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the error bar type for the series to saxlsXDirectionMinus
Series.ErrorBar = saxlsXDirectionMinus
```

Series.ErrorBarLine

Description

Returns a Line object representing the appearance of an ErrorBar.

Signature: VB.NET

```
Property ErrorBarLine As SALine (read-only)
```

Remarks

The properties of ErrorBarLine are LineColor, LineFormatting, LineStyle, and LineWeight.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set the error bar type for the series to saxlsXDirectionMinus
Series.ErrorBar = saxlsXDirectionMinus

'--- Format the error bar with the ErrorBarLine property
'--- ErrorBarLine returns SALine
'--- Make the line green
Series.ErrorBarLine.LineColor = RGB(0, 255, 0)
```

Series.HasSmoothedLine

Description

Sets or returns whether data points in a scatter chart are connected by smoothed lines.

Signature: VB.NET

```
Property HasSmoothedLine As Boolean (read/write)
```

Remarks

Valid only for scatter charts. When set to True, data points are connected by smoothed lines, that is, interpolated values, rather than real values, are displayed.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a scatter chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsScatterChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- For scatter charts, connect all data points with a smooth line
Series.HasSmoothedLine = True
```

Series.LabelFrame

Description

Returns a ChartFrame object representing the area of the series labels.

Signature: VB.NET

```
Property LabelFrame As IChartFrame (read-only)
```

Remarks

The properties of LabelFrame are Area, Border, HasShadow, Height, Text, TextFont, TextHorizontalAlignment, TextRotationAngle, TextVerticalAlignment, Width, X, and Y.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- LabelFrame returns an IChartFrame interface
'--- you can use to format
'--- the frame area around labels
'--- Set the LabelFrame ForegroundColor to red.
Series.LabelFrame.Area.ForegroundColor = RGB(255, 0, 0)
```

Series.LabelShowActualValue

Description

Sets or returns whether the series will include labels displaying actual Y values.

Signature: VB.NET

```
Property LabeLShowActualValue As Boolean (read/write)
```

Remarks

When set to True, the series will include labels displaying actual Y values. LabelShowActualValue is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Include labels that show actual Y axis values
Series.LabelShowActualValue = True
```

Series.LabelShowCategoryAndValueAsPercent

Description

Sets or returns whether a pie chart series will include labels displaying actual category (X) values and Y values as percents.

Signature: VB.NET

```
Property LabelShowCategoryAndValueAsPercent As Boolean (read/write)
```

Remarks

Property of a Pie chart. When set to True, the series will include labels displaying actual category (X) values, and Y values as percents. LabelShowCategoryAndValueAsPercent is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a pie chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsPieChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- For a Pie chart, show X and Y values as percents
Series.LabelShowCategoryAndValueAsPercent = True
```

Series.LabelShowCategoryLabel

Description

Sets or returns whether a pie chart series will include labels displaying actual category (X) values and Y values as percents.

Signature: VB.NET

```
Property LabelShowCategoryLabel As Boolean (read/write)
```

Remarks

Property of a Pie chart. When set to True, the series will include labels displaying actual category (X) values, and Y values as percents. LabelShowCategoryLabel is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a pie chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsPieChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- For Pie charts, include labels displaying actual
'--- category (X) values, and Y values as percents
Series.LabelShowCategoryLabel = True
```

Series.LabelShowValueAsPercent

Description

Sets or returns whether the series will include labels displaying Y values as percents.

Signature: VB.NET

```
Property LabelShowValueAsPercent As Boolean (read/write)
```

Remarks

When set to True, the series will include labels displaying Y values as percents. LabelShowValueAsPercent is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Show Y values as percents
Series.LabelShowValueAsPercent = True
```

Series.MarkerFormat

Description

Returns an Area object representing the data markers of a line, radar, or scatter series.

Signature: VB.NET

```
Property MarkerFormat As SAArea (read-only)
```

Remarks

The properties of MarkerFormat are AreaFormatting, BackgroundColor, ForegroundColor, and Pattern.

Setting the properties of MarkerFormat will have no effect unless you set MarkerFormat.AreaFormatting = ASXCreationType.saxlsCreationCustom

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a line chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsLineChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Setting the properties of MarkerFormat will have no
'--- effect unless the following is set:
Series.MarkerFormat.AreaFormatting = ASXCreationType.saxlsCreationCustom

'--- The MarkerFormat property returns an SAArea object
'--- that can be used
'--- to format the data markers of a chart
'--- (see description for applicable chart types)
'--- Set the foreground color to Blue
Series.MarkerFormat.ForegroundColor = RGB(0, 0, 255)
```

Series.MarkerSize

Description

Sets or retrieves the size of series data markers.

Signature: VB.NET

```
Property MarkerSize As Long (read/write)
```

Remarks

A data marker marks a single value in a series, corresponding to a single cell value. Line, radar, and scatter charts can include data markers.

Setting MarkerSize will have no effect unless you set MarkerFormat.AreaFormatting = ASXCreationType.saxlsCreationCustom

Series.MarkerType

Description

Sets or retrieves a data marker type for a series.

Signature: VB.NET

```
Property MarkerType As ASXMarkerType (read/write)
```

Remarks

A data marker marks a single value in a series, corresponding to a single cell value. Line, radar, and scatter charts may have data markers. ExcelWriter supports ten marker types, listed in the following table.

Setting MarkerType will have no effect unless you set MarkerFormat.AreaFormatting = ASXCreationType.saxlsCreationCustom

NoMarker	**0**
Square	**1**
Diamond	**2**
Triangle	**3**
X	**4**
Star	**5**
DowJones	**6**
StandardDeviation	**7**
Circle	**8**
PlusSign	**9**

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a line chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsLineChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Setting MarkerType will have no
'--- effect unless the following is set:
Series.MarkerFormat.AreaFormatting = ASXCreationType.saxlsCreationCustom

'--- Set the type of data marker to triangle
Series.MarkerType = 3
```

Series.Name

Description

Sets or returns the name of a single series.

Signature: VB.NET

```
Property Name As String (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Give a unique name to the series
Series.Name = "Sales Data"
```

Series.ScatterValues

Description

Sets or returns the range of cells containing the values of a scatter chart data series.

Signature: VB.NET

```
Property ScatterValues As String (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a scatter chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsScatterChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- For a scatter chart, assign the values
'--- in cells c1:c3 to the series
Series.ScatterValues = "C1:C3"
```

Series.ShowInLegend

Description

Sets or returns whether the specified series will appear in the chart's legend.

Signature: VB.NET

```
Property ShowInLegend As Boolean (read/write)
```

Remarks

Determines whether or not the specified series will appear in the chart's legend. ShowInLegend is set to True by default.

Examples

vbscript

```
series.ShowInLegend = False
```

Series.SpaceBetweenBars

Description

Sets or returns the space between the bars of the specified data series.

Signature: VB.NET

```
Property SpaceBetweenBars As Integer (read/write)
```

Remarks

SpaceBetweenBars is a Read/Write property. By default, SpaceBetweenBars is set to the value of Chart.SpaceBetweenBars.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Adjust the space between bars for the data series
Series.SpaceBetweenBars = 10
```

Series.SpaceBetweenCategories

Description

Sets or returns the space between space between the categories of the specified data series.

Signature: VB.NET

```
Property SpaceBetweenCategories As Integer (read/write)
```

Remarks

SpaceBetweenCategories is a Read/Write property. By default, SpaceBetweenCategories is set to the value of Chart.SpaceBetweenCategories.

Examples

vbscript

```
'--- Create an instance of ExcelWriter
'--- and add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Adjust the space between categories for the data series
Series.SpaceBetweenCategories = 10
```

Series.Values

Description

Sets or returns the range of cells from which to plot a data series.

Signature: VB.NET

```
Property Values As String (read/write)
```

Series.XErrorBarCustomMinusValues

Description

Sets or returns a custom minus error value for the X ErrorBar.

Signature: VB.NET

```
Property XErrorBarCustomMinusValues As String (read/write)
```

Remarks

To assign XErrorBarCustomMinusValues, set XErrorBarValueSource to 4 (saxlsCustom).

> ⚠ Only Scatter charts will display X error bars.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- First, set XErrorBarValueSource to saxlsCustom
Series.XErrorBarValueSource = saxlsCustom

'--- Now set the custom minus value
Series.XErrorBarCustomMinusValues = 5
```

Series.XErrorBarCustomPlusValues

Description

Sets or returns a custom plus error value for the X ErrorBar.

Signature: VB.NET

```
Property XErrorBarCustomPlusValues As String (read/write)
```

Remarks

To assign XErrorBarCustomPlusValues, set XErrorBarValueSource to 4 (saxlsCustom).

> ⚠ Only Scatter charts will display X error bars.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a scatter chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsScatterChart, _
    0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- First, set XErrorBarValueSource to saxlsCustom
Series.XErrorBarValueSource = saxlsCustom

'--- Now set the custom plus value
Series.XErrorBarCustomPlusValues = 5
```

Series.XErrorBarTShaped

Description

Sets or returns the shape of the X ErrorBar.

Signature: VB.NET

```
Property XErrorBarTShaped As Boolean (read/write)
```

Remarks

When XErrorBarTShaped is set to True, the X error bar will be T shaped. When XErrorBarTShaped is set to False, the X error bar will be a line. XErrorBarTShaped is True by default.

> ⚠ Only Scatter charts will display X error bars.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a scatter chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsScatterChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- XErrorBarTShaped is true by default
Series.XErrorBarTShaped = False
```

Series.XErrorBarValue

Description

Sets or returns the error value of an X ErrorBar.

Signature: VB.NET

```
Property XErrorBarValue As Double (read/write)
```

Remarks

XErrorBarValue may be a percentage, a fixed value, a standard deviation, a standard error, or a custom value, depending on the value of XErrorBarValueSource.

> ⚠ Only Scatter charts will display X error bars.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set XErrorBarValue to 15 percent
Series.XErrorBarValueSource = saxlsPercentage
Series.XErrorBarValue = 15
```

Series.XErrorBarValueSource

Description

Sets or returns the type of error value that the X ErrorBar will display.

Signature: VB.NET

```
Property XErrorBarValueSource As ASXErrorBarValueSource(read/write)
```

Remarks

XErrorBarValueSource may be any of the following.

saxlsPercentage	**1**	A percentage of each data point
saxlsFixedValue	**2**	A fixed value for all data points
saxlsStandardDeviation	**3**	A standard deviation from the series mean
saxlsCustom	**4**	A custom value
saxlsStandardError	**5**	The standard error of the data series

To assign a custom value, set XErrorBarCustomMinusValues and/or XErrorBarCustomPlusValues. To assign values to all other error types, set XErrorBarValue.

> ⚠ Only Scatter charts will display X error bars.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a scatter chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsScatterChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- First, set XErrorBarValueSource to saxlsCustom
Series.XErrorBarValueSource = saxlsCustom

'--- Now set the custom plus value
Series.XErrorBarCustomPlusValues = 5
```

Series.YErrorBarCustomMinusValues

Description

Sets or returns a custom minus error value for the Y ErrorBar.

Signature: VB.NET

```
Property YErrorBarCustomMinusValues As String (read/write)
```

Remarks

To assign YErrorBarCustomMinusValues, set YErrorBarValueSource to 4 (saxlsCustom).

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- First, set YErrorBarValueSource to saxlsCustom
Series.YErrorBarValueSource = saxlsCustom

'--- Now set the custom minus value
Series.YErrorBarCustomMinusValues = 5
```

Series.YErrorBarCustomPlusValues

Description

Sets or returns a custom plus error value for the Y ErrorBar.

Signature: VB.NET

```
Property YErrorBarCustomPlusValues As String (read/write)
```

Remarks

To assign YErrorBarCustomPlusValues, set YErrorBarValueSource to 4 (saxlsCustom).

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- First, set XErrorBarValueSource to saxlsCustom
Series.XErrorBarValueSource = saxlsCustom

'--- Now set the custom plus value
Series.XErrorBarCustomPlusValues = 5
```

Series.YErrorBarTShaped

Description

Sets or returns the shape of the Y ErrorBar.

Signature: VB.NET

```
Property YErrorBarTShaped As Boolean (read/write)
```

Remarks

When YErrorBarTShaped is set to True, the Y error bar will be T shaped. When YErrorBarTShaped is set to False, the Y error bar will be a line. YErrorBarTShaped is True by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a scatter chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsScatterChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- YErrorBarTShaped is true by default
Series.YErrorBarTShaped = False
```

Series.YErrorBarValue

Description

Sets or returns the error value of a Y ErrorBar.

Signature: VB.NET

```
Property YErrorBarValue As Double
```

Remarks

YErrorBarValue may be a percentage, a fixed value, a standard deviation, a standard error, or a custom value, depending on the value of YErrorBarValueSource.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set YErrorBarValue to 15 percent
Series.YErrorBarValueSource = saxlsPercentage
Series.YErrorBarValue = 15
```

Series.YErrorBarValueSource

Description

Sets or returns the type of error value that the Y ErrorBar will display.

Signature: VB.NET

```
Property YErrorBarValueSource As ASXErrorBarValueSource (read/write)
```

Remarks

YErrorBarValueSource may be any of the following values:

saxlsPercentage	**1**	A percentage of each data point
saxlsFixedValue	**2**	A fixed value for all data points
saxlsStandardDeviation	**3**	A standard deviation from the series mean
saxlsCustom	**4**	A custom value
saxlsStandardError	**5**	The standard error of the data series

To assign a custom value, set YErrorBarCustomMinusValues and/or YErrorBarCustomPlusValues. To assign values to all other error types, set YErrorBarValue.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- add a column chart to worksheet #1
Dim XlwApp, WrkSht, MyChart, Series
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 1, 1, 10, 10)

'--- Add a series to the series collection
Set Series = MyChart.SeriesCollection.Add("D2:E5", , 0)
MyChart.SeriesCollection.CategoryData = "{Monday, Tuesday}"

'--- Set YErrorBarValue to 15 percent
Series.YErrorBarValueSource = saxlsPercentage
Series.YErrorBarValue = 15
```

SeriesCollection

Description

The collection of data series in a single chart.

Signature: VB.NET

```
Public Class SeriesCollection
```

Remarks

To create a SeriesCollection object, use Chart.SeriesCollection.

Examples

vbscript

```
Set SeriesCollection = chart.Seriescollection
```

Properties

Name	Description
CategoryData	Sets or returns the range of Category Axis values.
Item	Returns a Series from the specified index of the SeriesCollection object.
LabelShowActualValue	Sets or returns whether the Y values of the Series will be shown.
LabelShowCategoryAndValueAsPercent	Sets or returns whether a pie chart will include series labels displaying actual category values and Y values as percents.
LabelShowCategoryLabel	Sets or returns whether to include labels displaying actual category values.

LabelShowValueAsPercent	Sets or returns whether to include labels on a pie chart displaying Y values as percents

Methods

Name	Description
Add(Area, , Long)	Adds a series to a chart.

SeriesCollection.Add(Area, , Long)

Description

Adds a series to a chart.

Signature: VB.NET

```
Function Add(Area As String, _
[After = -1], _
[Vertical As Long]) As ISeries
```

Parameters

Area

Specifies the values from which to plot the data series. Area may be a range of cells (such as "a1:c3"), or a sequence of values (such as "{5, 7, 9}").

After

(deprecated) Not used in current version.

Vertical

(optional) Specifies whether to plot the series from a range of cell values by row or by column. When Vertical is set to 0, ExcelWriter will plot the series by row. When Vertical is set to 1, ExcelWriter will plot the series by column. The default value of Vertical is 0.

Returns

A new Series object

Remarks

If you specify the optional parameter Vertical, include two commas between Area and Vertical, for a second parameter not used in the current version.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Add a Series to the collection.  The data will be
'--- from the range A1:C3.
MyChart.SeriesCollection.Add "A1:C3", , 0

'--- You can also add a Series by specifying the values in code
MyChart.SeriesCollection.Add "{1, 2, 3, 4}", , 0
```

SeriesCollection.CategoryData

Description

Sets or returns the range of Category Axis values.

Signature: VB.NET

```
Property CategoryData As String (read/write)
```

Remarks

Specifies the range of Category Axis (x-axis) values. CategoryData may be assigned as a range of cells ("A1:C3") or as a sequence of values ("{5,7,9}").

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Add a Series to the collection.  The data will be
'--- from the range A1:C3.
MyChart.SeriesCollection.Add "A1:C3", , 0

'--- Now set the Category Data (X axis)
'--- You can also set CategoryData with explicit
'--- values in code: CategoryData = "{April,May,June}"
MyChart.SeriesCollection.CategoryData = "A1:A3"
```

SeriesCollection.Item

Description

Returns a Series from the specified index of the SeriesCollection object.

Signature: VB.NET

```
Property Item(Index) As ISeries
```

Parameters

Index

Numeric index of the Series to return

Remarks

Represents a single series in a SeriesCollection. Item is the default property of SeriesCollection, so SeriesCollection(1) is equivalent to Charts.Item(1).

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart, SeriesOne, SeriesTwo
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Use the default Item property to get
'--- individual ISeries objects
'--- from the SeriesCollection.  This collection is 1-based
Set SeriesOne = MyChart.SeriesCollection(1)
Set SeriesTwo = MyChart.SeriesCollection(2)
```

SeriesCollection.LabelShowActualValue

Description

Sets or returns whether the Y values of the Series will be shown.

Signature: VB.NET

```
Property LabelShowActualValue As Boolean (read/write)
```

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set LabelShowActualValue to True to show series Y values
MyChart.SeriesCollection.LabelShowActualValues = True
```

SeriesCollection.LabelShowCategoryAndValueAsPercent

Description

Sets or returns whether a pie chart will include series labels displaying actual category values and Y values as percents.

Signature: VB.NET

```
Property LabelShowCategoryAndValueAsPercent As Boolean (read/write)
```

Remarks

Property of a Pie chart. When set to True, the chart will include series labels displaying actual category (X) values, and Y values as percents. LabelShowCategoryAndValueAsPercent is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set LabelShowCategoryAndValueAsPercent to True
'--- to show values as percentages
MyChart.SeriesCollection.LabelShowCategoryAndValueAsPercent = True
```

SeriesCollection.LabelShowCategoryLabel

Description

Sets or returns whether to include labels displaying actual category values.

Signature: VB.NET

```
Property LabelShowCategoryLabel As Boolean (read/write)
```

Remarks

When set to True, the chart will include series labels displaying actual category values. LabelShowCategoryLabel is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new clustered column chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsColumnChart, _
   0, 2, 2, 10, 20)

'--- Set LabelShowCategoryLabel to True
'--- to include series value labels
MyChart.SeriesCollection.LabelShowCategoryLabel = True
```

SeriesCollection.LabelShowValueAsPercent

Description

Sets or returns whether to include labels on a pie chart displaying Y values as percents

Signature: VB.NET

```
Property LabelShowValueAsPercent As Boolean (read/write)
```

Remarks

Property of a Pie chart. When set to True, the chart will include labels displaying Y values as percents. LabelShowValueAsPercent is False by default.

Examples

vbscript

```
'--- Create an instance of ExcelWriter and
'--- create a new pie chart
Dim XlwApp, WrkSht, MyChart
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)
Set MyChart = WrkSht.Charts.Add(saxlsPieChart, _
   0, 2, 2, 10, 20)

'--- For a Pie chart, set LabelShowValueAsPercent to True
'--- to ishow values as percentages of the whole
MyChart.SeriesCollection.LabelShowValueAsPercent = True
```

Style

Description

Represents a text style that can be assigned to one cell or a range of cells.

Signature: VB.NET

```
Public Class Style
```

Examples

vbscript

```
'--- To define a style, first set a Style object, and assign values to any or all of its properties
Dim mystyle
Set mystyle = xlw.CreateStyle
mystyle.Font.Name = "Arial"
mystyle.Font.Bold = True
mystyle.HorizontalAlignment = sahaRight

'--- Next, assign your style to an individual cell
ws.Cells("A1").Style = mystyle

'--- Alternatively, you could assign your style to a pre-defined Range object.
'--- For example, one that encompasses two rows and spans 5 columns:
Set Rng1 = Ws.Cells.Range(3, 3, 2, 5)
Rng1.style = mystyle

'--- Then, if you want to supplement the style for a single cell (for example,
'--- one that is contained within "Rng1") with additional formatting properties,
'--- use the Cell.Format property
Ws.Cells(3, 5).Format.WrapText = True
```

Properties

Name	Description

BackgroundColor	Sets or retrieves a style's background color.
BorderColor	Sets or retrieves a cell's border color.
BorderLineStyle	Sets or retrieves a cell's border style.
ForegroundColor	Sets or retrieves a style's foreground color.
Font	Sets or returns the font for your style.
Hidden	Sets or returns whether to hide cell formulas.
HorizontalAlignment	Sets or returns the horizontal alignmnet of the text in your spreadsheet.
Indent	Sets or returns the width of indentation of the contents within the cell as a number of character widths.
Locked	Sets or returns whether a cell can be modified.
Name	Sets or returns a style name.
Number	Sets or returns the display format of numbers and dates.
Orientation	Sets or returns the orientation of the text.
Pattern	Sets or returns a cell background pattern.
ShrinkToFit	Sets or returns whether text in a cell should be reduced in size to fit within the cell's width.
VerticalAlignment	Sets or returns the vertical alignment of the text in your spreadsheet.
WrapText	Sets or returns whether to wrap text within cells.

Style.BackgroundColor

Description

Sets or retrieves a style's background color.

Signature: VB.NET

```
Property BackgroundColor As Long (read/write)
```

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Create a SAStyle object
Set BlueStyle = XlwApp.CreateStyle

'--- Set the Background color
BlueStyle.BackgroundColor = RGB(0, 0, 255) ' blue

'--- Apply the style
Cells("A1").Value = "Blue Cell"
Cells("A1").Style = BlueStyle
```

Style.BorderColor

Description

Sets or retrieves a cell's border color.

Signature: VB.NET

```
Property BorderColor([Borders As SACellBorderIndex]) As Long (read/write)
```

Parameters

Borders

SACellBorderIndex value indicating the border for which you wish to change the color

Remarks

Use RGB values to assign colors, as in the example below.

BorderColor takes one parameters: an SACellBorderIndex value.

SACellBorderIndex	Values
sabpLeft	1
sabpRight	2
sabpTop	4
sabpBottom	8
sabpOutline (default)	15
sabpDiagonalDown	16
sabpDiagonalUp	32
sabpDiagonalBoth	48
sabpAll	63

Examples

vbscript

```
Set XlwApp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Create a SAStyle object
Set BorderStyle = XlwApp.CreateStyle()

'--- Set the Border type and color
BorderStyle.BorderColor(sabpOutline) = RGB(0, 0, 255) ' Blue

'--- Apply the style
Cells(1, 1).Style = BorderStyle
```

Style.BorderLineStyle

Description

Sets or retrieves a cell's border style.

Signature: VB.NET

```
Property BorderLineStyle([Borders As SACellBorderIndex]) As SABorderLineStyle (read/write)
```

Parameters

Borders

Remarks

BorderLineStyle takes an SACellBorderIndex as a parameter and can be set to an SABorderLineStyle value.

SABorderLineStyle	Values
sadgNone	0
sadgThin	1
sadgMedium	2
sadgDashed	3
sadgDotted	4
sadgThick	5
sadgDouble	6
sadgHair	7
sadgMediumDashed	8
sadgDashDot	9
sadgMediumDashDot	10
sadgDashDotDot	11
sadgMediumDashDotDot	12
sadgSlantedDashDot	13

SACellBorderIndex	Values
sabpLeft	1
sabpRight	2

sabpTop	**4**
sabpBottom	**8**
sabpOutline (default)	**15**
sabpDiagonalDown	**16**
sabpDiagonalUp	**32**
sabpDiagonalBoth	**48**
sabpAll	**63**

Examples

vbscript

```
Set XlwApp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Create a SAStyle object
Set BorderStyle = XlwApp.CreateStyle()

'--- Set the Border line style
BorderStyle.BorderLineStyle(sabpLeft) = sadgThin

'--- Apply the style
Cells(1, 1).Style = BorderStyle
```

Style.Font

Description

Sets or returns the font for your style.

Signature: VB.NET

```
Property Font As SAFont (read/write)
```

Remarks

Use the Font property to set a font for your style. Font contains several properties, each of which define an aspect your font. These properties are Name, Size, Bold, Italic, Underline, Strikethrough, Weight, Charset, Subscript, Superscript, and Color.

Examples

vbscript

```
Set XlwApp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Create a SAStyle object
Set BlueStyle = XlwApp.CreateStyle

'--- Set properties of the style's Font property
Set MyStyle = XlwApp.CreateStyle()
MyStyle.Font.Name = "Tahoma"
MyStyle.Font.Size = 10
MyStyle.Font.Italic = True

'--- Apply the style
Cells("a1").Style = MyStyle
```

Style.ForegroundColor

Description

Sets or retrieves a style's foreground color.

Signature: VB.NET

```
Property ForegroundColor As Long (read/write)
```

Remarks

A foreground color is visible when a pattern is assigned to a cell.

Examples

vbscript

```
Set XlwApp  = Server.CreateObject("SoftArtisans.ExcelWriter")
Set Cells = XlwApp.Worksheets(1).Cells

'--- Create a SAStyle object
Set BlueStyle = XlwApp.CreateStyle

'--- Set the Foreground color
BlueStyle.ForegroundColor = RGB(0, 0, 255) ' blue

'--- Apply the style
Cells("A1").value = "Blue text"
Cells("A1").style = BlueStyle
```

Style.Hidden

Description

Sets or returns whether to hide cell formulas.

Signature: VB.NET

```
Property Hidden As Boolean (read/write)
```

Remarks

Hidden hides cell formulas. The Hidden property is set to False by default.

Hiding cells has no effect unless the worksheet is protected. To protect a worksheet:

1. Open the worksheet in Microsoft Excel.
2. Open the Tools menu, and select Protection -> Protect Sheet.

Examples

vbscript

```
MyStyle.Hidden = True
```

Style.HorizontalAlignment

Description

Sets or returns the horizontal alignmnet of the text in your spreadsheet.

Signature: VB.NET

```
Property HorizontalAlignment As SACellHorizontalAlignment (read/write)
```

Remarks

HorizontalAlignment determines the horizontal alignment of the text in your spreadsheet. Assign a value to HorizontalAlignment by its name or number:

sahaUnspecified	**-1**
sahaGeneral	**0**
sahaLeft	**1**
sahaCenter	**2**
sahaRight	**3**
sahaFill	**4**
sahaJustify	**5**
sahaCenterAcross	**6**

Examples

vbscript

```
mystyle.HorizontalAlignment = sahaCenter
```

Style.Indent

Description

Sets or returns the width of indentation of the contents within the cell as a number of character widths.

Signature: VB.NET

```
Property Indent As Boolean (read/write)
```

Remarks

Set Indent to a value between 0 and 15.

Examples

vbscript

```
Set xlw = CreateObject("Softartisans.ExcelWriter")
Set ws = xlw.Worksheets(1)

ws.Cells(1, 1).Value = "This is indented text"

'--- Set indentation width to 3 character widths.
ws.Cells(1, 1).Format.Indent = 3

xlw.Save "c:\folder\out.xls", 0, 8
```

Style.Locked

Description

Sets or returns whether a cell can be modified.

Signature: VB.NET

```
Property Locked As Boolean (read/write)
```

Remarks

Locked determines whether a cell can be modified or not. The property is set to True by default.

Locking cells has no effect unless the worksheet is protected. To protect a worksheet:

1. Open the worksheet in Microsoft Excel.
2. Open the Tools menu, and select Protection -> Protect Sheet.

Or use Worksheet.WriteProtect.

Examples

vbscript

```
mystyle.Locked = False
```

Style.Name

Description

Sets or returns a style name.

Signature: VB.NET

```
Property Name As String (read/write)
```

Examples

vbscript

```
mystyle.Name = "MyStyleName"
```

Style.Number

Description

Sets or returns the display format of numbers and dates.

Signature: VB.NET

```
Property Number as String (read/write)
```

Remarks

For a complete list of Number codes, see Formatting Codes.

Examples

vbscript

```
mystyle.Number = 10
```

Style.Orientation

Description

Sets or returns the orientation of the text.

Signature: VB.NET

```
Property Orientation As SACellTextOrientation (read/write)
```

Remarks

Set Orientation to an SAOrientation value by name or number:

SAOrientation	Values	
saoriUnspecified	-1	Value not specified
saoriNoRotation	0	Default value
saoriTopToBottom	1	Displays text from top to bottom of the cell
saoriCounterclockwise	2	Rotates text 90 degrees counterclockwise
saoriClockwise	3	Rotates text 90 degrees clockwise
saoriCounterclockwise15	4	Rotates text 15 degrees counterclockwise
saoriClockwise15	5	Rotates text 15 degrees clockwise
saoriCounterclockwise30	6	Rotates text 30 degrees counterclockwise
saoriClockwise30	7	Rotates text 30 degrees clockwise
saoriCounterClockwise45	8	Rotates text 45 degrees counterclockwise
saoriClockwise45	9	Rotates text 45 degrees clockwise
saoriCounterclockwise60	10	Rotates text 60 degrees counterclockwise
saoriClockwise60	11	Rotates text 60 degrees clockwise
saoriCounterclockwise75	12	Rotates text 75 degrees counterclockwise
saoriClockwise75	13	Rotates text 75 degrees clockwise

Examples

vbscript

```
mystyle.Orientation = saoriClockWise
```

Style.Pattern

Description

Sets or returns a cell background pattern.

Signature: VB.NET

```
Property SACellBackgroundPattern As SACellBackgroundPattern (read/write)
```

Remarks

ExcelWriter supports the following 18 patterns:

saptrnSolid	1

saptrn50Gray	2
saptrn75Gray	3
saptrn25Gray	4
saptrnHorizontalStripe	5
saptrnVerticalStripe	6
saptrnReverseDiagonalStripe	7
saptrnDiagonalStripe	8
saptrnDiagonalCrosshatch	9
saptrnThickDiagonalCrosshatch	10
saptrnThinHorizontalStripe	11
saptrnThinVerticalStripe	12
saptrnThinReverseDiagonalStripe	13
saptrnThinDiagonalStripe	14
saptrnThinHorizontalCrosshatch	15
saptrnThinDiagonalCrosshatch	16
saptrn12Gray	17
saptrn6Gray	18

Examples

vbscript

```
Set mystyle = xlw.CreateStyle
mystyle.Pattern = 9
Cells("A1").value = "Diagonal Crosshatch"
Cells("A1").style = mystyle
```

Style.ShrinkToFit

Description

Sets or returns whether text in a cell should be reduced in size to fit within the cell's width.

Signature: VB.NET

```
Property ShrinkToFit As Boolean (read/write)
```

Examples

vbscript

```
Set xlw = CreateObject("Softartisans.ExcelWriter")
Set ws = xlw.Worksheets(1)

ws.Cells(3, 3).Value = "This text will be shrunk to fit in this cell"
ws.Cells(3, 3).Format.ShrinkToFit = True
ws.Cells.ColumnWidth(3) = 21

xlw.Save "c:\folder\out.xls", 0, 8
```

Style.VerticalAlignment

Description

Sets or returns the vertical alignment of the text in your spreadsheet.

Signature: VB.NET

```
Property VerticalAlignment As SACellVerticalAlignment (read/write)
```

Remarks

VerticalAlignment determines the vertical alignment of the text in your spreadsheet. VerticalAlignment has four possible values. You can assign a value by its name or number:

savaUnspecified	**-1**	Alignment is not specified
savaTop	**0**	Aligns text with the top of cells
savaCenter	**1**	Vertically centers text within cells
savaBottom	**2**	Aligns text with the bottom of cells (Default value)
savaJustify	**3**	Vertically justifies text within cells

Examples

vbscript

```
mystyle.VerticalAlignment = savaTop
```

Style.WrapText

Description

Sets or returns whether to wrap text within cells.

Signature: VB.NET

```
Property WrapText As Boolean (read/write)
```

Remarks

Use the WrapText property to wrap text within cells. WrapText may be either True or False. WrapText is set to False by default.

Examples

vbscript

```
mystyle.WrapText = True
```

Worksheet

Description

Represents a single Excel worksheet in a workbook.

Signature: VB.NET

```
Public Class Worksheet
```

Remarks

The first worksheet in an Excel workbook is WorkSheets(1), the second, WorkSheets(2), and so on.

Examples

vbscript

```
'--- Create the first worksheet in a workbook
Set ws = xlw.Worksheets(1)
```

Properties

Name	Description
Cells	Returns the set of cells contained in a worksheet.
Charts	Returns a Charts object representing the set of all charts in a single worksheet.
Name	Sets or returns the name of a worksheet.
PageSetup	Returns a PageSetup object representing the layout properties of a printed spreadsheet.
Pictures	Returns a Pictures object representing the set of all pictures in as single worksheet.
PivotTables	Returns the collection of PivotTables in a worksheet.
ShowGridlines	Sets or returns whether the worksheet will display gridlines.
ShowRowColHeaders	Sets or returns whether the worksheet will display row and column headers.
Visible	Sets or returns whether worksheets are visible, hidden, or very hidden.
WriteProtect	Sets or returns whether the worksheet is write protected.
Zoom	Sets or returns the zoom of the spreadsheet as a percent.

Methods

Name	Description
Copy()	The Copy method returns a new worksheet that is a copy of the original worksheet.
FreezePanes(SACell)	Freezes panes in a worksheet, allowing you to scroll through one pane while keeping other data visible.

SelectSheet()	Selects a worksheet to display when the workbook is opened.

Worksheet.Cells

Description

Returns the set of cells contained in a worksheet.

Signature: VB.NET

```
Property Cells As SACells (read-only)
```

Remarks

Cells represents the set of cells contained in a single worksheet. Individual Cells can be addressed by Name (for example, "B1") or by Number (for example, (1, 2)). References by name are easier to read, and match the cell references in Microsoft Excel. Accessing cells by number is significantly faster, and it is easier to iterate over a set of cells by number than by name. When referencing a cell by number, the format is cells(row, column).

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Use the Cells collection to access individual cells
WrkSht.Cells(1, 1).Value = "Cell A1"
WrkSht.Cells("A2").Value = "Cell A2"
```

Worksheet.Charts

Description

Returns a Charts object representing the set of all charts in a single worksheet.

Signature: VB.NET

```
Property Charts As ICharts (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Use the Charts object to work with Excel charts
'--- Get a reference to the first chart in the worksheet
Set MyChart = WrkSht.Charts(1)
```

Worksheet.Copy()

Description

The Copy method returns a new worksheet that is a copy of the original worksheet.

Signature: VB.NET

```
Function Copy() As SAWorksheet
```

Returns

A Worksheet identical to the current Worksheet excepting charts, image, or features that cannot be scripted by the ExcelApplication object.

Remarks

> This method does not copy charts, images, or features that cannot be scripted by the ExcelApplication object.

Examples

vbscript

```
Dim XlwApp, WrkSht, WrkShtCopy
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Make a copy of the first worksheet with the Copy() method
Set WrkShtCopy = WrkSht.Copy()
```

Worksheet.FreezePanes(SACell)

Description

Freezes panes in a worksheet, allowing you to scroll through one pane while keeping other data visible.

Signature: VB.NET

```
Sub FreezePanes(Cell As SACell)
```

Parameters

Cell

Cell at which to split the worksheet into panes.

Remarks

FreezePanes takes a cell and splits the worksheet above the cell, to the left of it, or both.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Horizontal split above row 20.
'--- Vertical split to the left of column H.
WrkSht.FreezePanes WrkSht.Cells("H20")

'--- Split worksheet vertically,
'--- to the left of column H, into two panes.
WrkSht.FreezePanes WrkSht.Cells("H1")
```

Worksheet.Name

Description

Sets or returns the name of a worksheet.

Signature: VB.NET

```
Property Name As String (read/write)
```

Remarks

Reference a worksheet by number, or by name. To reference a worksheet by name you must first assign a name to it using the Name property.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Name the first worksheet
WrkSht.Name = "First Worksheet"
```

Worksheet.PageSetup

Description

Returns a PageSetup object representing the layout properties of a printed spreadsheet.

Signature: VB.NET

```
Property PageSetup As SAPageSetup (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, PgSetup
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Get a reference to the SAPageSetup object for this worksheet
Set PgSetup = WrkSht.PageSetup
PgSetup.CenterHorizontally = True
```

Worksheet.Pictures

Description

Returns a Pictures object representing the set of all pictures in as single worksheet.

Signature: VB.NET

```
Property Pictures As SAPictures (read-only)
```

Examples

vbscript

```
Dim XlwApp, WrkSht, MyPic
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Use the Pictures object to work with pictures on the worksheet
Set MyPic = WrkSht.Pictures.CreateFromFile2(1, 1, "c:\pics\myPicture.jpg")
```

Worksheet.PivotTables

Description

Returns the collection of PivotTables in a worksheet.

Signature: VB.NET

```
Public ReadOnly Property PivotTables As IPivotTables
```

Remarks

A PivotTable is an interactive table that combines and compares large amounts of data. You can rotate its rows and columns to see different summaries of the source data, and you can display the details of different fields.

Examples

vbscript

```
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Open the workbook containing source data for the PivotTable.
XlwApp.Open "c:\reports\pivotdata.xls"

'--- Get a reference to the source data sheet.
Set wsData = XlwApp.Worksheets("DataSheet")

'--- Create a second worksheet and name it PivotSheet.
Set wsPivot = XlwApp.Worksheets(2)
wsPivot.Name = "PivotSheet"

'--- The data in the workbook is in a named range called
'--- DataNamedRange.  Get an SARange object representing
'--- that range.
Set rngData = XlwApp.NamedRange("DataNamedRange")

'--- Pass CreatePivotTable the range of source data to
'--- create the PivotTable.
Set pvtTable = wsPivot.PivotTables.CreatePivotTable(rngData, 2, 2)
```

Worksheet.SelectSheet()

Description

Selects a worksheet to display when the workbook is opened.

Signature: VB.NET

```
Sub SelectSheet()
```

Examples

vbscript

```
Dim XlwApp
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Select the third worksheet in the workbook
XlwApp.Worksheets(3).SelectSheet
```

Worksheet.ShowGridlines

Description

Sets or returns whether the worksheet will display gridlines.

Signature: VB.NET

```
Property ShowGridLines As Boolean
```

Remarks

When set to True, the worksheet will display gridlines. To hide gridlines, set ShowGridlines to False.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Turn off grid lines for the worksheet
WrkSht.ShowGridLines = False
```

Worksheet.ShowRowColHeaders

Description

Sets or returns whether the worksheet will display row and column headers.

Signature: VB.NET

```
Property ShowRowColHeaders As Boolean
```

Remarks

When set to True, the worksheet will display row and column headers. To hide row and column headers, set ShowRowColHeaders to False.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Turn off row and column headers for the worksheet
WrkSht.ShowRowColHeaders = False
```

Worksheet.Visible

Description

Sets or returns whether worksheets are visible, hidden, or very hidden.

Signature: VB.NET

```
Property Visible As SAXlSheetVisibility (read/write)
```

Remarks

You can set worksheets to be visible, hidden, or very hidden with ExcelWriter. When Visibility is set to true, Excel will display the worksheet, when set to false, Excel will hide the worksheet, but users can use the Unhide command to display the hidden worksheet. When set to "saxlSheetVeryHidden", users will need to use Visual Basic for Applications to unhide the hidden worksheet.

The Visible value names and codes are,

Constant name	VB Boolean Equivalent	Integer value
saxlSheetVisible	True	-1
saxlSheetHidden	False	0
saxlSheetVeryHidden		2

Please note: Excel will not open a workbook that has no visible sheets.

Examples

vbscript

```
Dim XlwApp
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Hide the first worksheet
XlwApp.Worksheets(1).Visible = saxlSheetHidden

'--- Make the second sheet "very hidden"
XlwApp.Worksheets(2).Visible = saxlSheetVeryHidden
```

Worksheet.WriteProtect

Description

Sets or returns whether the worksheet is write protected.

Signature: VB.NET

```
Property WriteProtect As Integer|Boolean (read/write)
```

Remarks

The worksheet for which this property is set will be write protected. This property is read/write. Users will be restricted from altering the formatting or layout of the worksheet, and they will be warned if they attempt to edit cell content. Users can turn off worksheet protection only if they supply the proper password in Excel.

To set write protection for a worksheet, set WriteProtect to an integer value. For example:

```
ws.WriteProtect = -13228
```

To remove write protection from a worksheet, set WriteProtect to False:

```
ws.WriteProtect = False
```

- The integer value you set the WriteProtect property to is not the password that you will use to unprotect a sheet in Excel. The integer value corresponds to a plaintext string password (for example, "-13228" is "pwd"). You must use the SAExcelApplication.Open method on an existing protected sheet to determine which integer value corresponds to your desired password. This behavior is by design. For a detailed explanation of this behavior, please see Features In Depth: Protecting your Data.
- This functionality provides write protection, not encryption, for a worksheet. Worksheet data will be fully readable when opened, but will be uneditable within Excel. This property should not be used to "protect" or "hide" sensitive data.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Write protect the worksheet
WrkSht.WriteProtect = -13228

'--- Remove write protection by setting False
WrkSht.WriteProtect = False
```

Worksheet.Zoom

Description

Sets or returns the zoom of the spreadsheet as a percent.

Signature: VB.NET

```
Property Zoom As Integer (read/write)
```

Remarks

Use the Zoom property to specify how large or small you want your document to appear as a percentage of its normal size. Zoom values are between 10 and 400; by default, Zoom is set to 100.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")
Set WrkSht = XlwApp.Worksheets(1)

'--- Set the worksheet zoom to 50% of its original size
WrkSht.Zoom = 50
```

Worksheets

Description

The collection of worksheets in a workbook.

Signature: VB.NET

```
Public Class Worksheets
```

Properties

Name	Description
Count	Returns the number of Worksheets in your workbook.

Indexers

Item	Returns the worksheet with the specified name or index.

Methods

Name	Description
Add(String)	Adds a worksheet to your Excel application.
Remove(Variant)	Removes a worksheet from the specified Worksheets collection.

Worksheets.Add(String)

Description

Adds a worksheet to your Excel application.

Signature: VB.NET

```
Function Add([SheetName As String]) As Integer
```

Parameters

SheetName

(optional) Name of the new Worksheet

Returns

Number of worksheets in the Worksheets collection as an integer

Examples

vbscript

```
xlw.Worksheets.Add
Dim Worksheet2
Set Worksheet2 = xlw.Worksheets(2)

xlw.Worksheets.Add
Dim Worksheet3
Set Worksheet3 = xlw.Worksheets(3)
```

Worksheets.Count

Description

Returns the number of Worksheets in your workbook.

Signature: VB.NET

```
Property Count As Integer (read-only)
```

Remarks

Counts the number of worksheets in your Excel application (workbook).

Examples

vbscript

```
Dim SheetCount, XlwApp
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Get the number of sheets in the workbook
SheetCount = XlwApp.Worksheets.Count
```

Worksheets.Item

Description

Returns the worksheet with the specified name or index.

Signature: VB.NET

```
Property Item As SAWorksheet (read-only)
```

Parameters

Worksheet

A string representing the name of the worksheet to be returned or a number representing the index of the worksheet to be returned

Remarks

Represents a single worksheet in a Worksheet collection. Item is the default property of Worksheets, so Worksheets(1) is equivalent to Worksheets.Item(1) . You can reference a Worksheets item either by number or by name.

Examples

vbscript

```
Dim XlwApp, WrkSht
Set XlwApp = Server.CreateObject("SoftArtisans.ExcelWriter")

'--- Get the first worksheet in the workbook
Set WrkSht = XlwApp.Worksheets(1)
```

Worksheets.Remove(Variant)

Description

Removes a worksheet from the specified Worksheets collection.

Signature: VB.NET

```
Sub Remove(Sheet As Variant)
```

Parameters

Sheet

Name or numeric index of the Worksheet to remove from the Worksheets collection.

Examples

vbscript

```
Set oXLW = Server.CreateObject("SoftArtisans.ExcelWriter")
oXLW.Worksheets.Remove(2)
```

Formula Functions

ExcelWriter supports almost all major math, string, boolean, time, statistical, database, lookup and reference formulas or functions that are part of Excel.

With ExcelWriter, formulas cannot be executed server-side to retrieve values for use in the script. All formulas are executed at the time that the file is downloaded and opened in Excel. (Please note: If the downloaded files are opened client side with the free Microsoft Excel Viewer, all formulas will be rendered with "0" as the resultant value.)

Formulas can be inserted into any cell in a worksheet by using the Cell.Formula property. They can be inserted either by hard coding the cell address or by dynamic coding using a variable, such as NumRows (the return value of the CopyFromRecordset method which indicates where the last row of the pasted recordset will be positioned in the worksheet) to provide a sum at the bottom of a column.

When applying a formula to a cell, always include the entire string in double quotation marks ("") and begin the string with an equal sign (=), as you would when creating a Microsoft Excel spreadsheet.

Any formula that is not mentioned in this list can be inserted into an existing spreadsheet or template and used in ExcelWriter. This list is comprised of formulas that can be added into your ExcelWriter script to be included at run-time.

The following example demonstrates using a formula.

```
ws.Cells("B6").Formula = "=SUM(B2:B3)"
```

ExcelWriter 2 allows you to use values from a different sheet when assigning a formula to a cell, as in the following examples.

```
ws3.Cells(1, 1).Formula = "=sum(sheet1\!A1,Sheet2\!A2)"
ws3.Cells(2, 1).Formula = "=sum(sheet1\!A1:A3)"
```

ExcelWriter does not support the following formulas:

- Sheet1:Sheet2!A1:C3
- Sheet1!A1:C3;Sheet4!A1:C3

ExcelWriter Supported Functions

Functions by Category	Functions Alphabetically
	• abs • acos • and • asin • atan • atan2 • average • ceiling

- Math Functions
  - abs
  - acos
  - asin
  - atan
  - atan2
  - ceiling
  - cos
  - degrees
  - even
  - exp
  - fact
  - floor
  - int
  - ln
  - log
  - log10
  - mod
  - odd
  - pi
  - product
  - radians
  - round
  - rounddown
  - roundup
  - sin
  - sqrt
  - subtotal
  - sum
  - sumif
  - tan
- String Functions
  - concatenate
  - len
  - mid
  - replace
  - value
- Logical Functions
  - and
  - false
  - if
  - not
  - or
  - true
- Time Functions
  - date
  - datevalue
  - day
  - days360
  - hour
  - minute
  - month
  - now
  - second
  - time
  - timevalue
  - today
  - weekday
  - year
- Statistical Functions
  - average
  - count
  - counta
  - countif
  - max
  - median
  - min
  - mode
  - percentile
  - stdev
  - stdevp

- Column
- Columns
- concatenate
- cos
- count
- counta
- countif
- date
- datevalue
- Daverage
- day
- days360
- Dcount
- Dcounta
- degrees
- Dget
- Dmax
- Dmin
- Dproduct
- Dstdev
- Dstdevp
- Dsum
- Dvar
- Dvarp
- even
- exp
- fact
- false
- floor
- fv
- hour
- if
- int
- iserr
- iserror
- isnumber
- len
- ln
- log
- log10
- max
- median
- mid
- min
- minute
- mod
- mode
- month
- not
- now
- npv
- odd
- or
- percentile
- pi
- pmt
- product
- pv
- radians
- replace
- round
- rounddown
- roundup
- Row
- Rows
- second
- sin
- sqrt
- stdev
- stdevp
- subtotal
- sum
- sumif
- tan
- time

- Information Functions
  - iserr
  - iserror
  - isnumber
  - type
- Lookup and Reference Functions
  - Column
  - Columns
  - Hlookup
  - Row
  - Rows
  - Vlookup
- Database Functions
  - Daverage
  - Dcount
  - Dcounta
  - Dget
  - Dmax
  - Dmin
  - Dproduct
  - Dstdev
  - Dstdevp
  - Dsum
  - Dvar
  - Dvarp
- Financial Functions
  - fv
  - npv
  - pmt
  - pv
- timevalue
- today
- true
- type
- value
- weekday
- year

Database Functions

- Daverage
- Dcount
- Dcounta
- Dget
- Dmax
- Dmin
- Dproduct
- Dstdev
- Dstdevp
- Dsum
- Dvar
- Dvarp

The function examples are based on the following database.

	A	B	C	D	E
1					
2					
3	**Last Name**	**First Name**	**Department**	**Salary**	**Age**
4					
5	Black	Helen	Marketing	60000	39
6	Brown	James	Development	42000	24
7	Gold	Elaine	Human Resources	43000	33
8	Green	Oliver	Marketing	59000	35
9	Silver	Louise	Development	55000	30
10					
11	**Department**	**Salary**			
12	Marketing	<60000			
13					
14	**Age**	**Age**			
15	>30	<30			
16					
17	**Department**	**Salary**			
18	Development	<55000			

Daverage

Returns the average of the values in a list or database, that meet a specified condition. Daverage takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field
  The column containing the values to average. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that cell values must meet to be included in the list of values to average. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display 54000, the average of all salaries of employees over 30.

```
ws.Cells("B30").Formula = "=Daverage(A3:E9,""Salary"",A14:A15)"
```

The embedded string "Salary" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dcount

Returns the number of cells in a column or database containing values that meet specified criteria. Dcount takes three arguments:

- Database
  A range of cells containing the database or list of values.
- [Field]
  The column containing the values to count. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.). Optional. If you do not specify Field, Dcount will count all database values that meet the specified conditions.
- Criteria
  A range of cells containing the conditions that cell values must meet to be included in the list of values to count. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value 3, the number of employees over 30.

```
ws.Cells("B30").Formula = "=Dcount(A3:E9,""Age"",A14:A15)"
```

The embedded string "Age" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dcounta

Returns the number of nonblank cells in a column or database containing values that meet specified criteria. Dcounta takes three arguments:

- Database
  A range of cells containing the database or list of values.
- [Field]
  The column containing the values to count. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.). Optional. If you do not specify Field, Dcount will count all database values that meet the specified conditions.
- Criteria
  A range of cells containing the conditions that cell values must meet to be included in the list of values to count. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value 4, the number of salaries below 60000.

```
ws.Cells("B30").Formula = "=Dcounta(A3:E9,""Salary"",B11:B12)"
```

The embedded string "Salary" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dget

Returns a single column value that meets specified conditions. If Dget finds more than one matching value, it returns the error #NUM!. If no matching value exists, Dget returns the error #VALUE!. Dget takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field
  The column containing the value to get. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that a cell value must meet. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value "Development," the only department containing an employee under age 30.

```
ws.Cells("B30").Formula = "=Dget(A3:E9,""Department"",B14:B15)"
```

The embedded string "Department" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dmax

Returns the highest value in a column that meets specified conditions. Dmax takes three arguments:
*Database
A range of cells containing the database or list of values.

- Field
  The column to search for maximum value. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that the cell value must meet. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value 55000, the highest salary in the Development department.

```
ws.Cells("B30").Formula = "=Dmax(A3:E9,""Salary"",A17:A18)"
```

The embedded string "Salary" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dmin

Returns the minimum value in a column that meets specified conditions. Dmin takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field&
  The column to search for minimum value. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that the cell value must meet. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value 59000, the lowest salary in the Marketing department.

```
ws.Cells("B30").Formula = "=Dmin(A3:E9,""Marketing"",A11:A12)"
```

The embedded string "Marketing" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dproduct

Returns the product of column values that meet specified conditions. Dproduct takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field
  The column containing the values to multiply. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that cell values must meet to be multiplied. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value 792, the product of 24 and 33, the ages of employees with salaries under 55000.

```
ws.Cells("B30").Formula = "=Dmax(A3:E9,""Age"",B17:B18)"
```

The embedded string "Age" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dstdev

Returns standard deviation based on a sample, of column values that meet a specified condition. The standard deviation is a measure of how widely values are dispersed from the average value. Dstdev takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field
  The column containing the values from which Dstdev will estimate standard deviation. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that cell values must meet to be included in the list of values to average. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value 3.055050463, the standard deviation from the average age of employees over 30, based on a sample.

```
ws.Cells("B30").Formula = "=Dstdev(A3:E9,""Age"",A14:A15)"
```

The embedded string "Age" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dstdevp

Returns standard deviation based on the entire population, of column values that meet a specified condition. The standard deviation is a measure of how widely values are dispersed from the average value. Dstdevp takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field
  The column containing the values from which Dstdevp will estimate standard deviation. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that cell values must meet to be included in the list of values to average. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value 2.494438258, the standard deviation from the average age of employees over 30, based on the entire population of arguments.

```
ws.Cells("B30").Formula = "=Dstdevp(A3:E9,""Age"",A14:A15)"
```

The embedded string "Age" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dsum

Returns the sum of all column values that meet a specified condition. Dsum takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field
  The column containing the values to sum. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that cell values must meet to be included in the list of values to sum. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display the value 85000, the sum of all salaries under 55000.

```
ws.Cells("B30").Formula = "=Dsum(A3:E9,""Salary"",B17:B18)"
```

The embedded string "Salary" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dvar

Returns the estimated variance in a population of column values that meet specified conditions, based on a sample. Dvar takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field
  The column containing the values for which to estimate variance. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that cell values must meet to be included in the list of values for which to estimate variance. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display 9.33333333, the estimated variance in ages over 30, assuming that the database contains only a sample of the population of employees.

```
ws.Cells("B30").Formula = "=Dvar(A3:E9,""Age"",A14:A15)"
```

The embedded string "Age" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Dvarp

Returns the variance in a population of column values that meet specified conditions, based on the entire population. Dvarp takes three arguments:

- Database
  A range of cells containing the database or list of values.
- Field
  The column containing the values for which to calculate variance. To specify Field, use the column name, or its number (use 1 for the first column in the database, 2 for the second, etc.).
- Criteria
  A range of cells containing the conditions that cell values must meet to be included in the list of values for which to calculate variance. The range must contain at least one column label with one or more conditions in cells below it.
  Example: If A3:E9 contains the database above, the following will display 6.22222222, the true variance in ages over 30, assuming that the database contains the entire population of employees.

```
ws.Cells("B30").Formula = "=Dvarp(A3:E9,""Age"",A14:A15)"
```

The embedded string "Age" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes.

Financial Functions

- Fv

- Npv
- Pmt
- Pv

Financial Functions		
Fv	Returns the future value of an investment of fixed regular payments, at a fixed rate of interest. Fv takes five arguments: • Rate Fixed interest rate per payment period • Nper Number of payment periods • Pmt Fixed payment amount per period • [Pv] Present value of the investment. Optional. Default value = 0. • [Type] Payment timing. 1 = the beginning of the payment period; 0 = the end of the period. Optional. Default value = 0.	The following will display the future value of an investment at 6% interest compounded monthly (0.5%/month), of eighteen payments of $150, with an initial value of $1000, where payments are made at the beginning of each period. `ws.Cells("B2").Formula = "=FV(0.005,18,-150,-1000,1)"` Represent payments as negative numbers, and money received as positive.
Npv	Returns the net present value of an investment based on a discount rate and a series of future payments (negative values) and income (positive values). Npv takes at least two arguments: • Rate The rate of discount over the length of one period. • Value1 Value1, value2... up to 29 values. Payments and income, equally spaced in time and occurring at the end of each period. Be sure to enter payment and income values in the correct sequence because the order of the values is used to interpret the order of cash flows.	The following will display the net present value of an investment with an annual discount rate of 10%, in which you pay $10,000 one year from today and receive an annual income of $3,000, $4,200 and $6,800 in the three years that follow. `ws.Cells("B2").Formula = "=NPV(0.10, -10000, 3000, 4200, 6800)"`
Pmt	Returns the amount of each loan payment, where both payments and interest rate are constant. Pmt takes five arguments: • Rate The loan's fixed interest rate • Nper Number of loan payments • Pv The present value - or, principal - of the loan • [Fv] Future value of the loan, that is, the desired remaining balance after the last payment. Optional. Default value = 0. • [Type] Payment timing. 1 = the beginning of the payment period; 0 = the end of the period. Optional. Default value = 0.	The following will display the monthly payment on a $15000 loan, at 8% interest per year, to be paid off in 24 months, with payments made at the beginning of each period. `ws.Cells("B2").Formula = "=PMT(0.08/12,24,15000,0,1)"` Represent payments as negative numbers, and money received as positive.
Pv	Returns the present value of an investment, where both payments and interest rate are constant. Pv takes five arguments: • Rate Fixed interest rate per period • Nper Number of payment periods • [Pmt] Fixed payment amount per period. Optional. If you omit Pmt, include Fv. • [Fv] Future value of the investment, that is, the desired balance after the last payment. Optional. Default value = 0. If you omit Fv, include Pmt. • [Type] Payment timing. 1 = the beginning of the payment period; 0 = the end of the period. Optional. Default value = 0.	The following will display the cost of a 25 year insurance annuity, with payments of $1000 at the beginning of each month, earning 8% interest per year. `ws.Cells("B2").Formula = "=PV(0.08/12,12*25,1000,0,1)"`

Information Functions

- iserr
- iserror
- isnumber
- type

String Functions	
iserr	Checks the type of value and returns True if it is any error value except #N/A, and False if it is not.
iserror	Checks the type of value and returns True if it is any error value (#N/A, #VALUE!, #REF!, #DIV/0!, #NUM!, #NAME?, or #NULL!), and False if it is not.
isnumber	Checks the type of a value and returns True if it is a number, and False if it is not. For example, isnumber(18) returns True, and isnumber("Sunday") returns False. If the value in B7 is 12, the following will display "True" in cell B8. `ws.Cells("B8").Formula = "=isnumber(B7)"`
type	Returns the type of value. Use type when the behavior of another function depends on the type of value in a particular cell. Type takes one argument, **value**, which can be any Microsoft Excel value, such as a number, text, or a logical value. The following are possible types and their associated values: • **Number:** 1 • **Text:** 2 • **Logical Value:** 4 • **Error Value:** 16 • **Array:** 64

Logical Functions

- and
- false
- if
- not
- or
- true

Logical Functions	
and	Takes 1 to 30 logical arguments and returns True if **all** the arguments are true, and False if one or more of the arguments is false. The arguments must have True/False values. An argument may be an array or cell reference that contains True/False values. The following will display the value "FALSE" in cell H2, since one of the arguments is false. `ws.Cells("H2").Formula = "=and(1+2=7, 1+2=3)"`
false	Returns the logical value FALSE. This function takes no arguments. To display the value FALSE in cell B6, use the following. `ws.Cells("B6").Formula = "=false()"`

if	Returns one of two specified values, depending on whether a condition is true or false. If takes three arguments: IF(test, value_if_true, value_if_false) The following lines will display "Profit" in cell F17 if the sum of G11:G16 is greater than 500, and "Loss" is the sum is 500 or less.
	
	`ws.Cells("G17").Formula = "=sum(g11:g16)"`
	`ws.Cells("F17").Formula = "=if(G17 > 500,""Profit"",""Loss"")"`
	
	Note that the embedded strings "Profit" and "Loss" are within additional sets of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:
	
	`ws.Cells("F17").Formula = "=if(G17 > 500,'Profit','Loss')"`

not	Takes one logical argument, and returns TRUE if the argument evaluates to FALSE, and FALSE if the argument evaluates to TRUE. The following will display "TRUE" in cell I2. `ws.Cells("I2").Formula = "=not(2+2=5)"`
or	Takes 1 to 30 logical arguments and returns True if **any** of the arguments evaluates to True, and False if all the arguments evaluate to False. `ws.Cells("J2").Formula = "=or(1+2=7, 1+2=3)"`
true	Returns the logical value TRUE. This function takes no arguments. To display the value TRUE in cell B6, use the following. `ws.Cells("B6").Formula = "=true()"`

Lookup and Reference Functions

- Column
- Columns
- Hlookup
- Row
- Rows
- Vlookup

Logical Functions	

Column	Returns the column number of a specified cell, or the array of column numbers of a specified range of cells. If you do not specify a cell, Column will return the column number of the cell containing the Column formula:
	
	`'Formula in cell G2 ----------  Value displayed in G2`
	`ws.Cells("G2").Formula = "=column(F3)"  '6`
	`ws.Cells("G2").Formula = "=column()"  '7`
	`ws.Cells("G2").Formula = "=column(A1:D6)"  '{1;2;3;4}`

Columns	Returns the number of columns in a specified range of cells. For example, cells A1:E1 span 5 rows, so the following will assign the value 5 to cell G7. `ws.Cells("G7").Formula = "=columns(A1:E1)"`

Hlookup	Returns a value in a specified table column, based on the value in the first row of the column (presumably, the column name), and a row number. Hlookup takes four arguments: • lookup_value The comparison value in the first row of the table (in most cases, the column name). Use either a cell reference, or a cell value. • table_array A range of cells representing the table being searched. • row_index_num Number of the row to search. • [range_lookup] If True, Hlookup will search against the next largest value that is smaller than lookup_value. If False, Hlookup will return #N/A if it cannot find lookup_value. Optional. Default value = True. Use Hlookup when the comparison value is in the first row of the table (that is, when the comparison value is the first value in a column), and Vlookup when the comparison value is in the first column of the table. **Example:** If cells G7:G10 contain "Last Name", "Brown", "Green", and "Silver", cells H7:H10 contain "First Name", "Helen", "James", and "Louise", and cells I7:I10 contain "Salary", 60000, 58000, 52000, the following will return 58000, which is the value in the third row of the "Salary" column. `=Hlookup("Salary",G7:I10,3)`

Row	Returns the row number of a specified cell, or the array of row numbers of a specified range of cells. If you do not specify a cell, Row will return the row number of the cell containing the Row formula.
	
	`'Formula in cell g2  ----------  'Value displayed in g2`
	`ws.cells("g2").formula = "=row(f3)"  '3`
	`ws.cells("g2").formula = "=row()"  '2`
	`ws.cells("g2").formula = "=row(a1:a5)"  '{1;2;3;4;5}`

Rows	Returns the number of rows in a specified range of cells. For example, cells A1:A4 span 4 rows, so the following will assign the value 4 to cell G7. `ws.Cells("G7").Formula = "=rows(A1:A4)"`
Vlookup	Returns a value in a specified table row, based on the value in the first column of the row, and a column number. Vlookup takes four arguments: • lookup_value The comparison value in the first column of the table. Use either a cell reference, or a cell value. • table_array A range of cells representing the table being searched. • row_index_num Number of the column to search. • [range_lookup] If True, Vlookup will search against the next largest value that is smaller than lookup_value. If False, Vlookup will return #N/A if it cannot find lookup_value. Optional. Default value = True. Use Vlookup when the comparison value is in the first column of the table (that is, when the comparison value is the first value in a row). Use Hlookup when the comparison value is in the first row of the table. **Example:** If cells G7:G10 contain "Last Name", "Brown", "Green", and "Silver", cells H7:H10 contain "First Name", "Helen", "James", and "Louise", and cells I7:I10 contain "Salary", 60000, 58000, 52000, the following will return 60000, which is the value in the third column of Helen Brown's row. `=Vlookup("Brown",G7:I10,3)`

Math Functions

- abs
- acos
- asin
- atan
- atan2
- ceiling
- cos

- degrees
- even
- exp
- fact
- floor
- int
- ln
- log
- log10
- mod
- odd
- pi
- product
- radians
- round
- rounddown
- roundup
- sin
- sqrt
- subtotal
- sum
- sumif
- tan

Math Functions	
abs	Returns the absolute value of a number. In the following example, B8 will contain the absolute value of the number in cell B7. `ws.Cells("B8").Formula = "=abs(B7)"`

acos	Returns the arccosine, or inverse cosine, of a number. Acos takes one argument, **number**, which is the cosine of the angle returned by acos. Number must be a value from -1 to 1. The angle returned by acos is given in radians in the range 0 (zero) to pi. To convert the result from radians to degrees, multiply it by 180/pi(), or use the degrees function.
	In the following example, the arccosine is the angle whose cosine is 0.5.
	`ws.Cells("B8").Value = 0.5`
	`ws.Cells("B8").Formula = "=acos(B7)"`

asin	Returns the arcsine, or inverse sine, of a number. Asin takes one argument, **number**, which is the sine of the angle returned by asin. Number must be a value from -1 to 1. The angle returned by asin is given in radians in the range -pi/2 to pi/2. To convert the result from radians to degrees, multiply it by 180/pi(), or use the degrees function.
	In the following example, the arcsine is the angle whose sine is 0.3.
	`ws.Cells("B8").Value = 0.3`
	`ws.Cells("B8").Formula = "=asin(B7)"`

atan	Returns the arctangent, or inverse tangent, of a number. Atan takes one argument, **number**, which is the tangent of the angle returned by atan. The angle returned by atan is given in radians in the range -pi/2 to pi/2. To convert the result from radians to degrees, multiply it by 180/pi(), or use the degrees function.
	In the following example, the arctangent is the angle whose tangent is 1.
	`ws.Cells("B8").Value = 1`
	`ws.Cells("B8").Formula = "=atan(B7)"`

atan2	Returns the arctangent, or inverse tangent, of specified x and y coordinates. The arctangent is the angle from the x-axis to a line containing the origin (0, 0) and a point with coordinates (x_num, y_num). A positive atan2 result represents a counterclockwise angle from the x-axis; a negative result represents a clockwise angle. Atan2(a,b) equals atan(b/a), except that a can equal 0 in atan2. If both x_num and y_num are 0, atan2 returns the #DIV/0! error value. The angle returned by atan2 is given in radians between -pi and pi, excluding -pi. To convert the result from radians to degrees, multiply it by 180/pi(), or use the degrees function. The following returns the arctangent of the point 1, 1 (the angle between the x-axis and the line from the origin to point 1, 1) in degrees (45). `ws.Cells("B8").Formula = "=degrees(atan2(1,1))"`
ceiling	Returns a number rounded up, away from zero, to the nearest multiple of significance. For example, if you want to avoid using pennies in your prices and your product is priced at $4.42, use the formula ceiling(4.42,0.05) to round prices up to the nearest nickel. Ceiling takes two arguments, **Number** and **Significance**. Number is the value to round up, and Significance is the multiple to which you want to round. If number is an exact multiple of significance, no rounding occurs. If either argument is not a number, ceiling returns the #VALUE! error value. Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number and significance have different signs, ceiling returns the #NUM! error value. The following rounds 3.21 up to 3.25. `ws.Cells("B8").Formula = "=ceiling(3.21,0.05)"`
cos	Returns the cosine of a specified angle. Cos takes one argument, **Number**, which is the angle in radians for which to return the cosine. If the angle is in degrees, multiply it by pi()/180 or use the radians function to convert it to radians. `ws.Cells("B8").Formula = "=cos(radians(45))"`
degrees	Converts radians into degrees. Degrees takes one argument, **angle**, which is the angle in radians to convert. The following takes pi radians and returns 180 degrees. `ws.Cells("F5").Formula = "=degrees(pi())"`
even	Returns a number rounded up to the nearest even integer. Even takes one argument, **Number**, which the is the value to round up. You can use this function for processing items that come in twos. For example, a packing crate accepts rows of one or two items. The crate is full when the number of items, rounded up to the nearest two, matches the crate's capacity. If the value passed to even is not a number, the function returns the #VALUE! error value. Regardless of the sign of the Number argument, a value is rounded up when adjusted away from zero. If number is an even integer, no rounding occurs. The following displays 32 in cell f5. `ws.Cells("F5").Formula = "=even(31)"`
exp	Returns e raised to the power of the argument. The constant e equals approximately 2.71828182845904, the base of the natural logarithm. In the following example, B8 will contain B7 raised to the power of e. `ws.Cells("B8").Formula = "=exp(B7)"`
fact	Returns the factorial of a number. The factorial of a number is equal to 1*2*3*...* number. Fact takes one argument, **Number**, which is the non-negative number for which to return the factorial. If the argument is not an integer, it is truncated. The following displays 24 (1*2*3*4) in cell c5. `ws.Cells("C5").Formula = "=fact(4)"`

floor	Rounds a specified number down, toward zero, to the nearest multiple of significance. Floor takes two arguments, **Number** and **Significance**. Number is the value to round down. Significance is the multiple to which you want to round. If either argument is non-numeric, floor returns the #VALUE! error value. If Number and Significance have different signs, floor returns the #NUM! error value. Regardless of the sign of number, a value is rounded down when adjusted away from zero. If number is an exact multiple of significance, no rounding occurs. The following rounds 3.27 down to 3.25. `ws.Cells("B8").Formula = "=floor(3.27,0.05)"`
int	Rounds a number down to the nearest integer. In the following example, B8 will contain the value of B7, rounded down to the nearest integer. `ws.Cells("B8").Formula = "=int(B7)"`
ln	Returns the natural logarithm of a number. Natural logarithms are based on the constant e (approximately 2.71828182845904). Ln takes one argument, **Number**, which is the positive real number for which you want the natural logarithm. Ln is the inverse of exp. The following returns 4.276666, the natural logarithm of 72. `ws.Cells("B8").Formula = "=ln(72)"`
log	Returns the logarithm of a number to the specified base. Log takes two arguments, **Number** and **Base**. Number is the positive real number for which to return the logarithm. Base is the base of the logarithm. If base is omitted, it is assumed to be 10. The following returns 3, the logarithm of 8 with base 2 (3). `ws.Cells("B8").Formula = "=log(8,2)"`
log10	Returns the base-10 logarithm of a number. Log10 takes one argument, **Number**, which is the positive real number for which to return the base-10 logarithm. The following returns 1, the base-10 logarithm of 10. `ws.Cells("B8").Formula = "=log10(10)"`
mod	Returns the remainder after a number is divided. Mod takes two arguments, **Number** and **Divisor**. For example, mod(52,5) returns the remainder after dividing 52 by 5, that is, 2. To display this value in cell B6, use `ws.Cells("B6").Formula = "=mod(52, 5)"`
odd	Returns a number rounded up to the nearest odd integer. Odd takes one argument, **Number**, which is the value to round up. If the argument is non-numeric, odd returns the #VALUE! error value. Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an odd integer, no rounding occurs. The following displays 7 in cell F5. `ws.Cells("F5").Formula = "=odd(6)"`
pi	Returns the number 3.14159265358979, the mathematical constant pi, accurate to 15 digits.
product	Multiplies all the numbers given as arguments and returns the product. Product takes 1 to 30 numbers. Arguments that are numbers, logical values, or text representations of numbers are counted; arguments that are error values or text that cannot be translated into numbers cause errors. If an argument is an array or reference, only numbers in the array or reference are counted. Empty cells, logical values, text, or error values in the array or reference are ignored. The following multiplies the values in cells h12 to h20, and displays the product in cell h22. `ws.Cells("H22").Formula = "=product(H12:H20,5)"`

radians	Converts degrees to radians. Degrees takes one argument, angle, which is the angle in degrees to convert. The following takes 180 degrees and returns pi radians. `ws.Cells("F5").Formula = "=radians(180)"`
round	Rounds a number to a specified number of digits. In the following example, B6 will contain the value of B5, rounded to 2 decimal places. `ws.Cells("B6").Formula = "=round(B5, 2)"`
rounddown	Rounds a number down, toward zero. Rounddown takes two arguments **Number** and **Num_digits**. Number is the value to round down. Num_digits is the number of digits to which you want to round the number. If num_digits is greater than 0 (zero), then number is rounded down to the specified number of decimal places. If num_digits is 0, then number is rounded down to the nearest integer. If num_digits is less than 0, then number is rounded down to the left of the decimal point. The following will round 9.2745 down to 2 decimal places, and display 9.27 in cell F5. `ws.Cells("F5").Formula = "=rounddown(9.2745,2))"`
roundup	Rounds a number up, away from 0 (zero). Roundup takes two arguments **Number** and **Num_digits**. Number is the value to round up. Num_digits is the number of digits to which you want to round the number. If num_digits is greater than 0 (zero), then number is rounded up to the specified number of decimal places. If num_digits is 0, then number is rounded up to the nearest integer. If num_digits is less than 0, then number is rounded up to the left of the decimal point. The following will round 9.2745 up to 2 decimal places, and display 9.28 in cell F5. `ws.Cells("F5").Formula = "=roundup(9.2745,2))"`
sin	Returns the sine of the given angle. Sin takes one argument, **Number**, which is the angle in radians for which to return the sine. If the angle in degrees, multiply it by pi()/180 or use the radians function to convert it to radians. `ws.Cells("B8").Formula = "=sin(radians(45))"`
sqrt	Returns a positive square root. Sqrt takes one argument, **Number**, which is the number for which to return the square root. If Number is negative, sqrt returns the #NUM! error value. `ws.Cells("B8").Formula = "=sqrt(9)"`

subtotal	Returns the subtotal of a set of cells using a specified function, in a list or database. **Subtotal** takes as arguments **function_num** and 1 to 29 cell references or ranges. **Function_num** is a number from 1 to 11 that represents the function used to calculate the subtotal, as listed in the following table.
	
	• **average:** 1
	• **count:** 2
	• **counta:** 3
	• **max:** 4
	• **min::** 5
	• **product:** 6
	• **stdev:** 7
	• **stdevp:** 8
	• **sum:** 9
	• **var:** 10
	• **varp:** 11
	
	`'--- Generate a subtotal of B1:B4 using the average function`
	`ws.Cells("B5").Formula = "=subtotal(1,B1:B4)"`
	
	`'--- Generate a subtotal of A1:A4 using the sum function`
	`ws.Cells("A5").Formula = "=subtotal(9,A1:A4)" ||`
	`||{anchor:sum} sum | Adds all the numbers in a range of cells.  In the following example, the sum of the values in cells B2 through B5 will be displayed in cell B6. {code:vb.net}ws.Cells("B6").Formula = "=sum(B2:B5)"`

sumif	Adds a range of cells if a specified condition is met. **Sumif** takes three arguments: **range**, **criteria**, and **sum_range**. **Range** is the set of cells to be evaluated. **Criteria** is the condition to be met. **Sum_range** contains the cells to be summed if the corresponding cells in **range** meet the specified condition. For example, if range A1:A4 contains {32;45;67;23}, the condition is ">44," and sum_range B1:B4 contains {3;7;2;4}, only the second and third members of sum_range (7 and 2) will be summed, because only the second and third members of range (45 and 67) are greater than 44. To display the result of this example (9) in cell B5, use,
	
	`ws.Cells("B5").Formula = "=sumif(A1:A4,"">44"",B1:B4)"`
	
	Notice that the embedded string ">44" is within an additional set of double-quotes. ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:
	
	`ws.Cells("B5").Formula = "=sumif(A1:A4,'>44',B1:B4)"`

tan	Returns the tangent of the given angle. Tan takes one argument, **Number**, which is the angle in radians for which to return the tangent. If the angle in degrees, multiply it by pi()/180 or use the radians function to convert it to radians. `ws.Cells("B8").Formula = "=tan(radians(45))"`

Statistical Functions

- average
- count
- counta
- countif
- max
- median
- min
- mode
- percentile
- stdev
- stdevp

Statistical Functions

average	Returns the average (arithmetic mean) of the arguments. In the following example, B8 will contain the average of the values in cells B2 through B7. `ws.Cells("B8").Formula = "=average(B2:B7)"`
count	Counts the number of cells that contain numbers and numbers within the list of arguments. Arguments that are numbers, dates, or text representations of numbers are counted; arguments that are error values or text that cannot be translated into numbers are ignored. For example, if the values in cells A1:A7 are {"red","yellow", "seven", "blue", 11/2/1999, "green", 8}, count(A1:A7) would equal 3 ("seven", 11/2/1999, 8). To display this result in cell A8, use, `ws.Cells("A8").Formula = "=count(A1:A7)"`
counta	Counts the number of cells that are not empty and the values within a list of arguments. Counta takes up to 30 arguments, which may be either cells or values (not contained in cells). For example, if the values in cells A1:A3 are {January, *empty cell*, December}, count(A1:A3) would equal 2, and count(A1:A3, "Sunday") would equal 3. To display count(A1:A3) in cell A4, use: `ws.Cells("A4").Formula = "=counta(A1:A3)"`
countif	Counts the number of cells that contain values that meet a specified condition. Countif takes two arguments: • **Range:** The range of cells from which cells will be counted if their values meet the specified condition. • **Criteria:** The condition that a cell must meet to be counted. The condition may be a number, expression, or text. The following displays in cell G2 the number of cells within A1:D3 that contain values greater than 40. `ws.Cells("G2").Formula = "=countif(A1:D3,'>40')"`
max	Returns the largest value in a set of values. In the following example, B8 will contain the largest value appearing in cells B2 through B7. `ws.Cells("B8").Formula = "=max(B2:B7)"`
median	Returns the median value in a set of numbers. For example, if cells B1:B5 contain the numbers {82,65,19,6,301}, median(B1:B5) would return 65. To display this return value in cell B6, use: `ws.Cells("B6").Formula = "=median(B1:B5)"`
min	Returns the smallest value in a set of values. In the following example, B8 will contain the smallest value appearing in cells B2 through B7. `ws.Cells("B8").Formula = "=min(B2:B7)"`
mode	Returns the most frequently occurring, or repetitive, value in an array or range of data. Mode takes up to 30 arguments (number1, number2,...) arguments for which to calculate the mode. You can also use a single array or a reference to an array instead of arguments separated by commas. The arguments should be numbers, names, arrays, or references that contain numbers. If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included. If the data set contains no duplicate data points, mode returns the #N/A error value. The following returns the value that occurs most frequently in the cell range B6:B20. `ws.Cells("C13").Formula = "=mode(B6:B20)"`

percentile	Returns the k-th percentile of values in a range. You can use this function to establish a threshold of acceptance. For example, you can decide to examine candidates who score above the 90th percentile. Percentile takes two arguments **Array** and **K**. Array is the array or range of data that defines relative standing. K is the percentile value in the range 0 to 1, inclusive. If array is empty or contains more than 8,191 data points, percentile returns the #NUM! error value. If k is non-numeric, percentile returns the #VALUE! error value. If k is < 0 or if k > 1, percentile returns the #NUM! error value. `ws.Cells("H22").Formula = "=percentile(H12:H18,5)"`
stdev	Estimates standard deviation based on a sample. The standard deviation is a measure of how widely values are dispersed from the average value. To display the estimated standard deviation of the sample of values 12,34,45,56 in cell B6, use: `ws.Cells("B8").Formula = "=stdev(12,34,45,56)"`
stdevp	Calculates standard deviation based on the entire population. To display the standard deviation of the values 12,34,45,56 in cell B6, use: `ws.Cells("B6").Formula = "=stdevp(12,34,45,56)"`

String Functions

- concatenate
- len
- mid
- replace
- value

String Functions	
concatenate	Concatenates a series of up to thirty strings. If cell I31 contains Smith, J31 Margaret, and P31 contains her social security number, the following will display "Margaret Smith's social security number is 013-364-8625." `ws.Cells("B6").Formula = "=concatenate(J31,I31,"'s ","phone number is ",P31)"`
len	Returns the number of characters in a text string. In the following example, B6 will contain the number of characters in the string in B5. `ws.Cells("B6").Formula = "=len(B5)"`
mid	Returns a specific number of characters from a text string, starting at position you specify. In the following example, B6 will contain the two character text string starting at the third character in the string B5. `ws.Cells("B6").Formula = "=mid(B5, 3, 2)"`

replace	Replaces part of a text string with another text string. Replace takes four arguments: **Old_text**, **Start_num**,**Num_chars**, and **New_text**. **Old_text** is the original string. **Start_num** is the position of the character in the original string at which you want the replacement to begin. **Num_chars** is the number of original characters you want to replace. **New_text** is the string that will replace part of the original string. For example, the result of replace("Jane",2,3,"oan") is "Joan." To display this result in cell B6, use:
	`ws.Cells("B6").Formula = "=replace(""Jane"",2,3,""oan"")"`
	The embedded strings "Jane" and "oan" are within additional sets of double-quotes because ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:
	`ws.Cells("B6").Formula = "=replace('Jane',2,3,'oan')"`

value	Converts a string representing a number to a number. To convert the string "$32" to the number 32 use:
	`ws.Cells("B20").Formula = "=Value(""$32"")"`
	The embedded string "$32" is within an additional set of double-quotes because ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:
	`ws.Cells("B20").Formula = "=Value('$32')"`

Time Functions

- date
- datevalue
- day
- days360
- hour
- minute
- month
- now
- second
- time
- timevalue
- today
- weekday
- year

If start_date occurs after end_date, days360 returns a negative number. Dates should be entered by using the date function, or as results of other formulas or functions. For example, use date(2008,5,23) for the 23rd day of May, 2008.

The following returns the number of days between April 8, 2003 and July 19, 2003, based on a 360-day year.

```
ws.Cells("B4").Formula = "=Days360(date(2003,4,8),date(2003,7,19),true)"
```

Time Functions

date	Returns the number that represents the date in Microsoft Excel date-time code. **Date** takes three arguments: Year, Month, and Day. **Year** is a number from 1900 to 1999 in Microsoft Excel for Windows. **Month** is a number from 1 to 12 representing the month of the year. **Day** is a number from 1 to 31 representing the day of the month. To retrieve the Excel date-time code for Aug 9, 2000, and display it in cell B2, use: `ws.Cells("B2").Formula = "=Date(2000,8,9)"`
datevalue	Returns the Excel date-time code of the date represented by text. Datevalue takes one argument, **date_text**, which represents a date in an Excel date format. For example, "1/30/2008" or "30-Jan-2008" are text strings within quotation marks that represent dates. Using the default date system in Excel for Windows, date_text must represent a date from January 1, 1900, to December 31, 9999. Datevalue returns the #VALUE! error value if date_text is out of this range. If the year portion of date_text is omitted, datevalue uses the current year from your computer's built-in clock. Time information in date_text is ignored. Excel stores dates as sequential serial numbers so they can be used in calculations. By default, January 1, 1900 is serial number 1, and January 1, 2008 is serial number 39448 because it is 39,448 days after January 1, 1900. The following takes the date string "12/20/2003" and returns the date-time code 37975. `ws.Cells("B2").Formula = "=Datevalue("12/20/2003")`
day	Returns the day of the month (a number from 1 to 31) of a date represented in Excel date-time code. In the following example, 36747 is the Excel date-time code for Aug 9, 2000. To display the day of the month (9) in cell B2, use, `ws.Cells("B2").Formula = "=Day(36747)"`
days360	Returns the number of days between two dates based on a 360-day year (twelve 30-day months), which is used in some accounting calculations. Days360 takes three arguments, **start_date**, **end_date**, and **method**. Start_date and end_date are the two dates between which you want to know the number of days. Method is a logical value that specifies whether to use the U.S. or European method in the calculation: • **False or omitted:** U.S. (NASD) method. If the starting date is the 31st of a month, it becomes equal to the 30th of the same month. If the ending date is the 31st of a month and the starting date is earlier than the 30th of a month, the ending date becomes equal to the 1st of the next month; otherwise the ending date becomes equal to the 30th of the same month. • **True:** European method. Starting dates and ending dates that occur on the 31st of a month become equal to the 30th of the same month.

hour	Returns the hour of a time value. The hour returned is given as an integer ranging from 0 (12:00 A.M.) to 23 (11:00 P.M.). **Hour** takes one argument, a time-value. The time-value may be entered as:
	• A text string (for example, "6:45 PM")
	• A decimal number in Excel date-time code (for example, 0.78125, which represents 6:45 PM)
	To display the return value of **hour** in cell B4, use:
	`ws.Cells("B4").Formula = "=Hour(""6:45 PM"")"`
	The embedded string "6:45 PM" is within an additional set of double-quotes because ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:
	`ws.Cells("B4").Formula = "=Hour('6:45 PM')"`

minute

Returns the minute, an integer from 0 to 59, corresponding to a time value. The time-value may be entered as,

- A text string (for example, "31-Dec-00" or "12-31-00")
- A number in Excel date-time code (for example, 35886, which represents the date 4-1-98)
  To display the minute of the time value "4:48 PM" in cell B5, use

```
ws.Cells("B5").Formula = "=Minute(""4:48 PM"")"
```

The embedded string "4:48 PM" is within an additional set of double-quotes because ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:

```
ws.Cells("B5").Formula = "=Minute('4:48 PM')"
```

month

Returns the month, an integer from 1 (January) to 12 (December), corresponding to a date-value. The date-value may be entered as:

- A text string (for example, "31-Dec-00")
- A number in Excel date-time code (for example, 36891, which represents 31-Dec-00)
  To display the month of the date-value "31-Dec-00" in cell B5, use:

```
ws.Cells("B6").Formula = "=Month(""31-Dec-00"")"
```

The embedded string "31-Dec-00" is within an additional set of double-quotes because ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:

```
ws.Cells("B6").Formula = "=Month('31-Dec-00')"
```

now

Returns the time-value in Excel date-time code of the current date or time. This function takes no arguments. To display the current time in cell B7, use:

```
'--- Format.number defines the display format of numbers,
'--- dates, and times.  18 will display the time as h:mm AM/PM.
ws.Cells("B7").Format.Number = 18
ws.Cells("B7").Formula = "=Now()"
```

second

Returns the second, an integer in the range 0 to 59, corresponding to a time-value in Excel date-time code. The time-value may be entered as:

- A text string (for example, "13:50:25" or "1:50:25 PM")
- A number in Excel date-time code (for example, 0.576678241, which represents the time 1:50:25 PM)
  To display the second of the time value "1:50:25 PM" in cell B9, use:

```
ws.Cells("B9").Formula = "=Second(""13:50:25"")"
```

The embedded string "13:50:25" is within an additional set of double-quotes because ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:

```
ws.Cells("B9").Formula = "=Second('13:50:25')"
```

time	Returns a fraction from 0 (00:00:00) to 0.99999999 (23:59:59) that represents a particular time. **Time** takes three arguments: **hour** , **minute**, and **second**. To display the fraction code for the time 12:25:30 in cell B10, use: `ws.Cells("B10").Formula = "=Time(12,25,30)"`
timevalue	Returns the decimal number of the time represented by a text string. The decimal number is a value ranging from 0 (zero) to 0.99999999, representing the times from 0:00:00 (12:00:00 AM) to 23:59:59 (11:59:59 P.M.). Timevalue takes one argument, **time_text**, a text string that represents a time in any one of the Microsoft Excel time formats; for example, "6:45 PM" and "18:45" text strings within quotation marks that represent time. Date information in time_text is ignored. The following takes the time string "2:57 pm" and returns the decimal number 0.622917. `ws.Cells("B10").Formula = "=Timevalue("2:57 pm")`

today	Returns a Microsoft Excel date-time code value that represents today's date. The following example demonstrates displaying today's date in the format mm/dd/yyyy. For other display options, see Formatting Codes.
	
	`ws.Cells("B18").Format.Number = 14`
	`ws.Cells("B18").Formula = "=Today()"`

weekday	Returns a number from 1 to 7 identifying the day of the week corresponding to a value that represents a date. The date-value may be entered as:
	
	• A text string (for example, "31-Dec-00")
	• A number in Excel date-time code (for example, 36891, which represents 31-Dec-00)
	To display the day of the week for "31-Dec-00" in cell B5, use:
	
	`ws.Cells("B11").Formula = "=Weekday(""31-Dec-00"")"`
	
	The embedded string "31-Dec-00" is within an additional set of double-quotes because ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:
	
	`ws.Cells("B11").Formula = "=Weekday('31-Dec-00')"`

year	Returns the year, an integer in the range 1900-9999, corresponding to a date-value. The date-value may be entered as:
	
	• A text string (for example, "31-Dec-00")
	• A number in Excel date-time code (for example, 36891, which represents 31-Dec-00)
	To display the year for "31-Dec-00" in cell B5, use:
	
	`ws.Cells("B12").Formula = "=Year(""31-Dec-00"")"`
	
	The embedded string "31-Dec-00" is within an additional set of double-quotes because ASP syntax requires that double-quotes embedded in a string be doubled. Alternatively, use single quotes:
	
	`ws.Cells("B12").Formula = "=Year('31-Dec-00')"`

Formula Calculation Operators

ExcelWriter supports all but two of Microsoft Excel's operators. Use operators to perform calculations on formula elements.

Examples:

- The following SUMIF formula includes the Range operator (:), the Union operator (,), and the Greater Than operator (>).

```
ws.Cells("B5").Formula = "=SUMIF(A1:A4,'>44',B1:B4)"
```

- The following PV formula includes the Division operator (/), the Union operator (,), and the Multiplication operator (*).

```
ws.Cells("B2").Formula = "=PV(0.08/12,12*25,1000,0,1)"
```

ExcelWriter does **not** support,

- The Percent operator (%)
- The Intersection operator (single space)

Reference Operators

Range operator: (colon)	Creates a reference to the range of cells between two cell references. The range includes the two cells. **Example:** A1:C12
Union operator, (comma)	Creates a single reference or value from two or more references and/or values. **Example:** SUMIF(A1:A4,'>44',B1:B4)

Comparison Operators

Equal to = (equal sign)	The **Equal to** operator produces TRUE if the values compared are equal and FALSE if they are not. **Example:** C2=D5
Greater than > (greater than sign)	The **Greater than** operator produces TRUE if the first value is greater than the second and FALSE if it is not.**Example:** C2>D5
Less than< (less than sign)	The **Less than** operator produces TRUE if the first value is less than the second and FALSE if it is not. **Example:** C2<D5
Greater than or equal to>= (greater than or equal to sign)	The **Greater than or equal to** operator produces TRUE if the first value is greater than or equal to the second and FALSE if it is not.**Example:** C2>=D5
Less than or equal to<= (less than or equal to sign)	The **Less than or equal to** operator produces TRUE if the first value is less than or equal to the second and FALSE if it is not.**Example:** C2<=D5
Not equal to<> (not equal to sign)	The **Not equal to** operator produces TRUE if the compared values are not equal and FALSE if the compared values are equal.**Example:** C2<>D5

Arithmetic operators

Addition+ (plus sign)	**Example:** 8+2
Subtraction, Negation- (minus sign)	**Examples:** 12-2, -1
Multiplication* (asterisk)	**Example:** 5*7
Division/ (forward slash)	**Example:** 9/3
Exponentiation^ (caret)	**Example:** 2^3

Text Operator

Concatenation& (ampersand)	Concatenates two strings. **Example:** The result of ="jelly"&"bean" is "jellybean."

Formatting Codes

Use the Number property to set the display format of numbers and dates. Number may be a property of the Style object or of Cell.Format. The

table below lists predefined formats. You can also define custom formats, for example, a timestamp format:

```
Set TimeStampStyle = XLW.CreateStyle
TimeStampStyle.Number = "mm/dd/yyyy hh:mm:ss AM/PM"
```

A custom format may include four format code sections, separated by semicolons (. Excel reads the four sections as follows.

- Section 1 defines a format for positive numbers
- Section 2 defines a format for negative numbers
- Section 3 defines a format for zero values
- Section 4 defines a format for text

When you skip a section, include the empty section's ending semicolon. If you include less than four sections, in order, do not include a semicolon after the last section.

To make negative numbers red in a custom format, put [Red] in front of the code for the negative number. For example, to define a custom number style with black positive numbers and red negative numbers you would do this:

```
Set myStyle = XLW.CreateStyle
myStyle.Number = "0;[Red]0"
```

Format.Number Value	Format String
0	General
1	0
2	0.00
3	#,##0
4	#,##0.00
5	($#,##0_);($#,##0)
6	($#,##0_);($#,##0)
7	($#,##0.00_);($#,##0.00)
8	($#,##0.00_);($#,##0.00)
9	0%
10	0.00%
11	0.00E+00
12	# ?/?
13	# /
14	m/d/yy
15	0.00%
16	d-mmm
17	mmm-yy
18	h:mm AM/PM
19	h:mm:ss AM/PM
20	h:mm
37	(#,##0_);(#,##0)
38	(#,##0_);(#,##0)
39	(#,##0.00_);(#,##0.00)
40	(#,##0.00_);(#,##0.00)

41	_(* #,##0_);(* (#,##0);(* "-");(@_)
42	_($* #,##0_);($* (#,##0);(* "-");(@_)
43	_(* #,##0.00_);(* (#,##0.00);(* "-");(@_)
44	_($* #,##0.00_);($* (#,##0.00);(* "-");(@_)
45	mm:ss
46	[h]:mm:ss
47	mm:ss.0
48	##0.0E+0
49	@ (Format as text. The value will be displayed exactly as entered.)

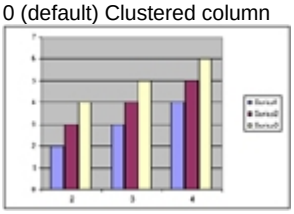
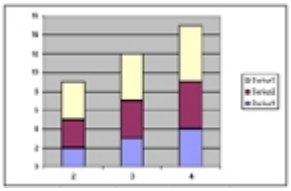
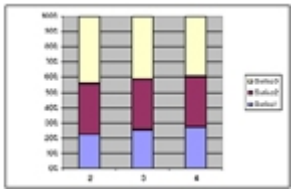
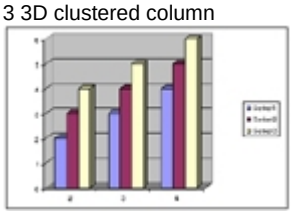
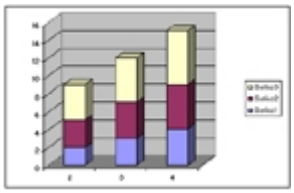
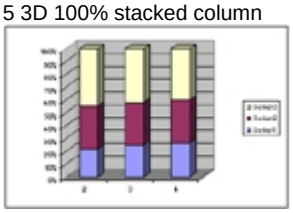
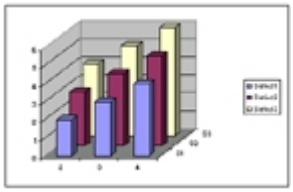
Chart Codes

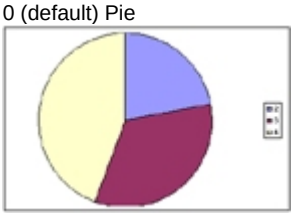
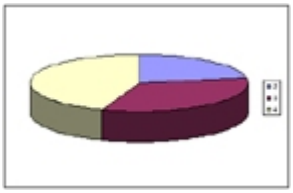
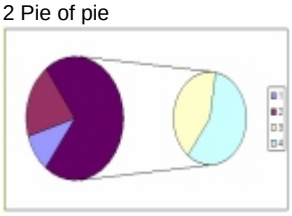
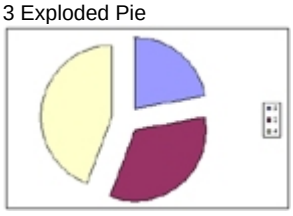
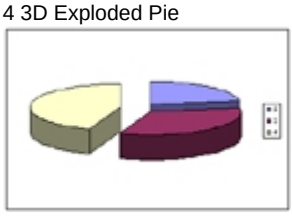
When you add a chart to your worksheet using the Add method, you must specify a chart type by name (e.g. saxlsCylinderChart) or number. You may also specify a subtype; if you do not, ExcelWriter will use the default subtype. ExcelWriter currently supports fifteen major Excel chart types and over seventy subtypes. The type and subtype codes are listed below.

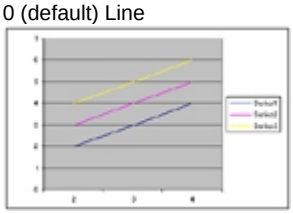
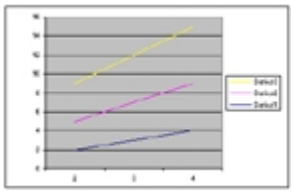
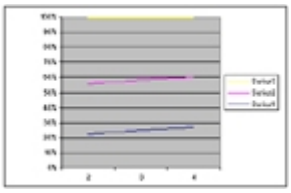
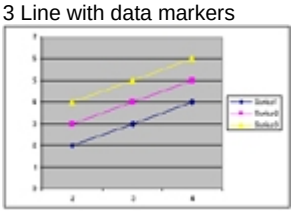
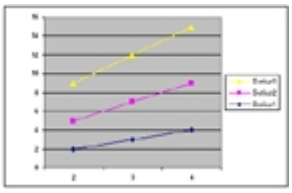
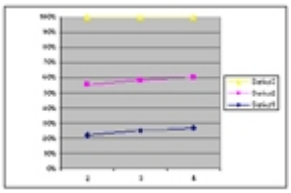
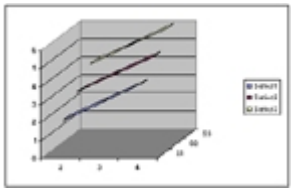
> ⚠ ExcelWriter is not a charting engine and cannot be used to produce static gif or jpeg representations of charts for inclusion in an HTML page.

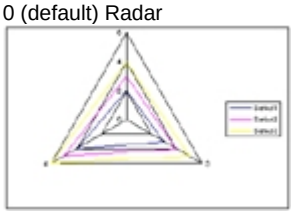
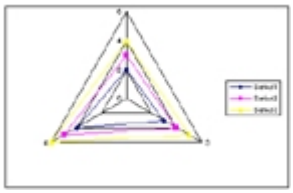
- Column
- Pie Chart
- Line Chart
- Radar Chart
- Bar Chart
- Surface Chart
- Area Chart
- Scatter Chart
- Doughnut Chart
- Bubble Chart
- Cylinder Chart
- Cone Chart
- Pyramid Chart
- Column Line Chart
- Column Line 2 Axes Chart

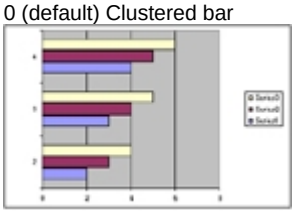
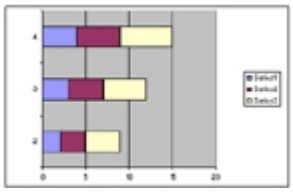
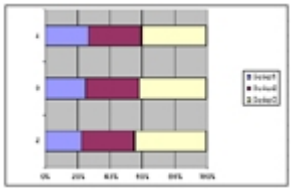
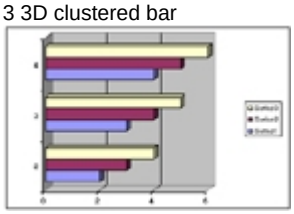
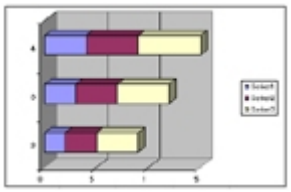
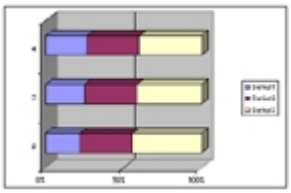
Chart types and Codes	Subtypes and Codes

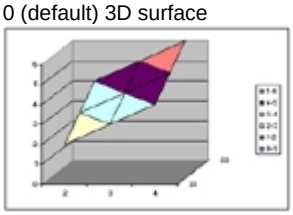
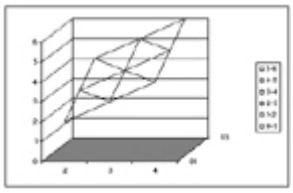
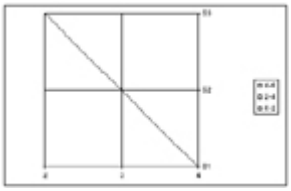
Column Chart **Name: saxIsColumnChart** **Number: 0**	• 0 (default) Clustered column  • 1 Stacked column  • 2 100% stacked column  • 3 3D clustered column  • 4 3D stacked column  • 5 3D 100% stacked column  • 6 3D column 

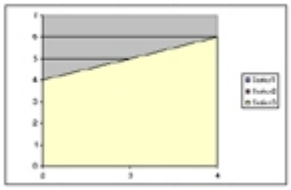
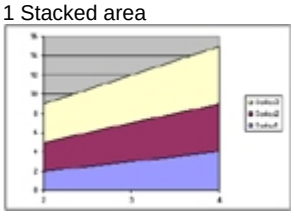
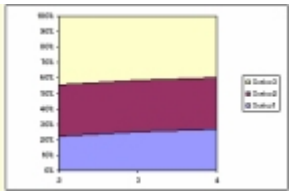
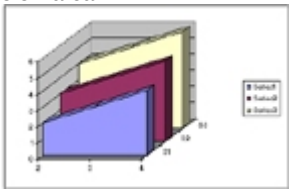
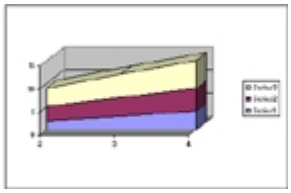
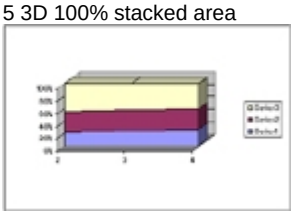
Pie Chart **Name: saxIsPieChart** **Number: 1**	• 0 (default) Pie  • 1 3D Pie  • 2 Pie of pie  • 3 Exploded Pie  • 4 3D Exploded Pie  • 5 Bar of pie

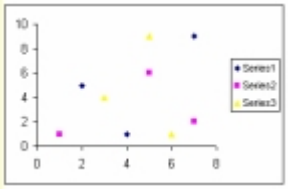
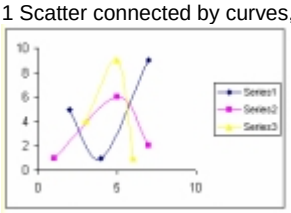
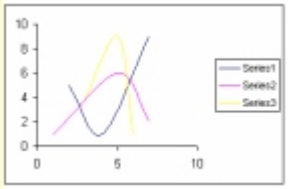
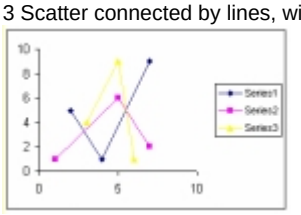
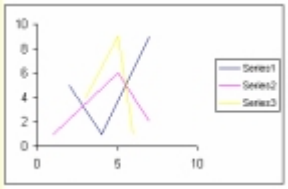
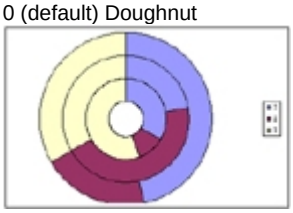
Line Chart **Name: saxlsLineChart** **Number: 2**	• 0 (default) Line  • 1 Stacked line  • 2 100% stacked line  • 3 Line with data markers  • 4 Stacked line with data markers  • 5 100% stacked line with data markers  • 6 3D line 

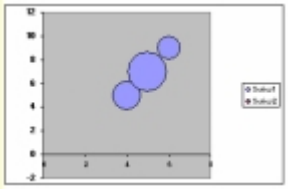
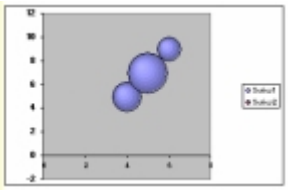
Radar Chart **Name: saxlsRadarChart** **Number: 3**	• 0 (default) Radar  • 1 Radar with data markers  • 2 Filled radar

Bar Chart **Name: saxIsBarChart** **Number: 4**	• 0 (default) Clustered bar  • 1 Stacked bar  • 2 100% stacked bar  • 3 3D clustered bar  • 4 3D stacked bar  • 5 3D 100% stacked bar 

Surface Chart **Name: saxIsSurfaceChart** **Number: 5**	• 0 (default) 3D surface  • 1 Wireframe 3D surface  • 2 Contour • 3 Wireframe contour 

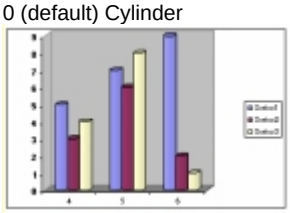
Area Chart **Name: saxIsAreaChart** **Number: 6**	• 0 (default) Area  • 1 Stacked area  • 2 100% stacked area  • 3 3D area  • 4 3D stacked area  • 5 3D 100% stacked area 

Scatter Chart **Name: saxIsScatterChart** **Number: 7**	• 0 (default) Scatter  • 1 Scatter connected by curves, with data markers  • 2 Scatter connected by curves, without data markers  • 3 Scatter connected by lines, with data markers  • 4 Scatter connected by lines, without data markers 
Doughnut Chart **Name: saxIsDoughnutChart** **Number: 8**	• 0 (default) Doughnut  • 1 Exploded doughnut

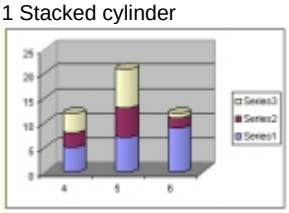
Bubble Chart **Name: saxlsBubbleChart** **Number: 9** ***This chart type not available in Excel 95 (BIFF7 format).***	• 0 (default) Bubble  • 1 3D bubble 

Cylinder Chart
Name: saxlsCylinderChart
Number: 10
This chart type not available in Excel 95 (BIFF7 format).

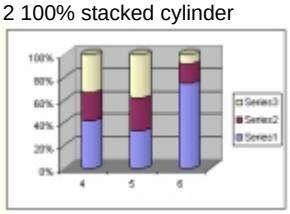
- 0 (default) Cylinder



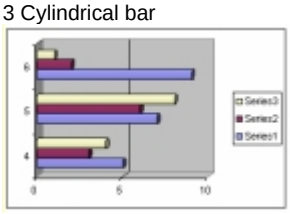
- 1 Stacked cylinder



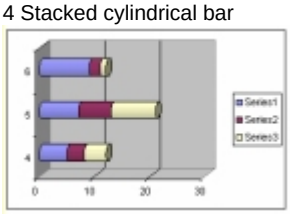
- 2 100% stacked cylinder



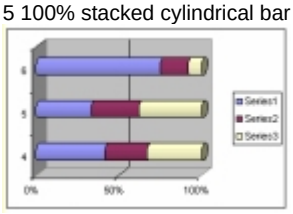
- 3 Cylindrical bar



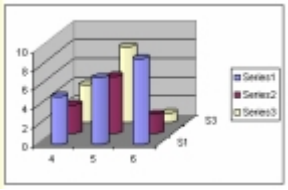
- 4 Stacked cylindrical bar



- 5 100% stacked cylindrical bar

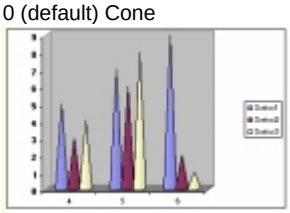


- 6 3D cylindrical column

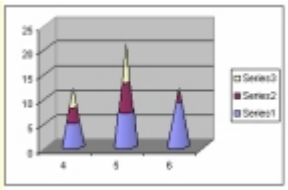


Cone Chart
Name: saxlsConeChart
Number: 11
This chart type not available in Excel 95 (BIFF7 format).

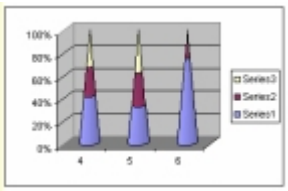
- 0 (default) Cone



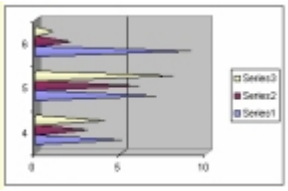
- 1 Stacked cone



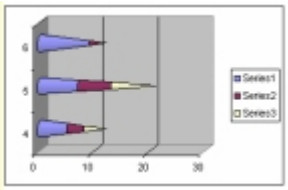
- 2 100% stacked cone



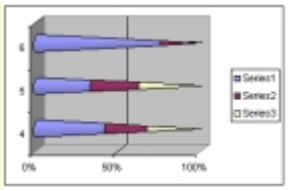
- 3 Conical bar



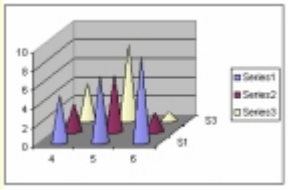
- 4 Stacked conical bar



- 5 100% stacked conical bar

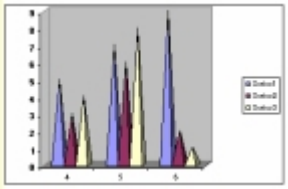


- 6 3D conical column

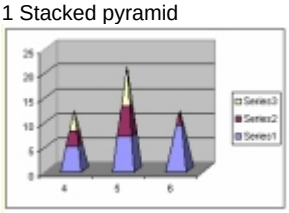


Pyramid Chart
Name: saxlsPyramidChart
Number: 12
This chart type not available in Excel 95 (BIFF7 format).

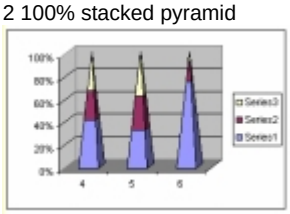
- 0 (default)



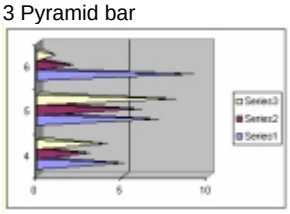
- 1 Stacked pyramid



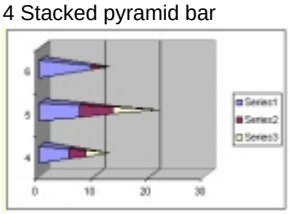
- 2 100% stacked pyramid



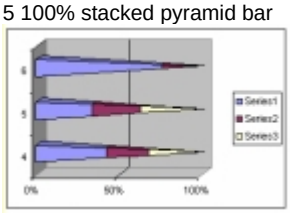
- 3 Pyramid bar



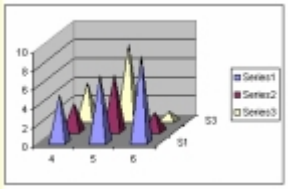
- 4 Stacked pyramid bar

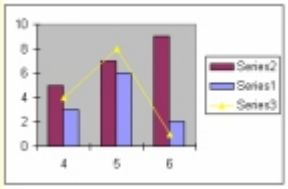
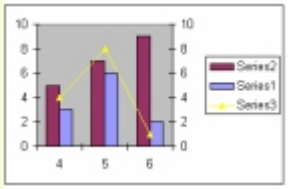


- 5 100% stacked pyramid bar



- 6 3D pyramid column



Column Line Chart **Name: saxIsColumnLineChart** **Number: 13**	• 0 (default)  • 1 Stacked • 2 100% stacked
Column Line 2 Axes Chart **Name: saxIsColumnLine2AxesChart** **Number: 14**	• 0 (default)  • 1 Stacked • 2 100% stacked

PivotTable AutoFormat Types

To apply one of Excel's built-in AutoFormat types to a PivotTable, use the PivotTable object's AutoFormatType property.

saxlAutoFormatClassic1

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatClassic2

Sum of Sales	Product								
Month	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	***Grand Total***
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatTable1

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatTable2

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatTable3

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	$ 110.00	$ 700.00	$ 650.00	$ 245.00	$ 780.00	$ 234.00	$ 780.00	$ 780.00	$ 4,279.00
2/1/2003	120.00	680.00	700.00	220.00	600.00	244.00	760.00	800.00	4,124.00
3/1/2003	130.00	660.00	750.00	195.00	300.00	254.00	740.00	820.00	3,849.00
4/1/2003	140.00	640.00	800.00	170.00	900.00	264.00	720.00	840.00	4,474.00
5/1/2003	150.00	620.00	850.00	145.00	300.00	274.00	700.00	860.00	3,899.00
6/1/2003	160.00	600.00	900.00	120.00	300.00	284.00	680.00	880.00	3,924.00
7/1/2003	170.00	580.00	950.00	95.00	500.00	294.00	660.00	900.00	4,149.00
8/1/2003	180.00	560.00	1,000.00	70.00	780.00	304.00	640.00	920.00	4,454.00
9/1/2003	190.00	540.00	1,050.00	45.00	456.00	314.00	620.00	940.00	4,155.00
10/1/2003	200.00	520.00	1,100.00	20.00	345.00	324.00	600.00	960.00	4,069.00
11/1/2003	210.00	500.00	1,150.00	20.00	278.00	334.00	580.00	980.00	4,052.00
12/1/2003	220.00	480.00	1,200.00	25.00	34.00	344.00	560.00	1,000.00	3,863.00
Grand Total	$ 1,980.00	$ 7,080.00	$11,100.00	$ 1,370.00	$ 5,573.00	$ 3,468.00	$ 8,040.00	$10,680.00	$49,291.00

saxlAutoFormatTable4

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	$ 110.00	$ 700.00	$ 650.00	$ 245.00	$ 780.00	$ 234.00	$ 780.00	$ 780.00	$ 4,279.00
2/1/2003	120.00	680.00	700.00	220.00	600.00	244.00	760.00	800.00	4,124.00
3/1/2003	130.00	660.00	750.00	195.00	300.00	254.00	740.00	820.00	3,849.00
4/1/2003	140.00	640.00	800.00	170.00	900.00	264.00	720.00	840.00	4,474.00
5/1/2003	150.00	620.00	850.00	145.00	300.00	274.00	700.00	860.00	3,899.00
6/1/2003	160.00	600.00	900.00	120.00	300.00	284.00	680.00	880.00	3,924.00
7/1/2003	170.00	580.00	950.00	95.00	500.00	294.00	660.00	900.00	4,149.00
8/1/2003	180.00	560.00	1,000.00	70.00	780.00	304.00	640.00	920.00	4,454.00
9/1/2003	190.00	540.00	1,050.00	45.00	456.00	314.00	620.00	940.00	4,155.00
10/1/2003	200.00	520.00	1,100.00	20.00	345.00	324.00	600.00	960.00	4,069.00
11/1/2003	210.00	500.00	1,150.00	20.00	278.00	334.00	580.00	980.00	4,052.00
12/1/2003	220.00	480.00	1,200.00	25.00	34.00	344.00	560.00	1,000.00	3,863.00
Grand Total	$ 1,980.00	$ 7,080.00	$11,100.00	$ 1,370.00	$ 5,573.00	$ 3,468.00	$ 8,040.00	$10,680.00	$49,291.00

saxlAutoFormatTable5

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	$ 110.00	$ 700.00	$ 650.00	$ 245.00	$ 780.00	$ 234.00	$ 780.00	$ 780.00	$ 4,279.00
2/1/2003	120.00	680.00	700.00	220.00	600.00	244.00	760.00	800.00	4,124.00
3/1/2003	130.00	660.00	750.00	195.00	300.00	254.00	740.00	820.00	3,849.00
4/1/2003	140.00	640.00	800.00	170.00	900.00	264.00	720.00	840.00	4,474.00
5/1/2003	150.00	620.00	850.00	145.00	300.00	274.00	700.00	860.00	3,899.00
6/1/2003	160.00	600.00	900.00	120.00	300.00	284.00	680.00	880.00	3,924.00
7/1/2003	170.00	580.00	950.00	95.00	500.00	294.00	660.00	900.00	4,149.00
8/1/2003	180.00	560.00	1,000.00	70.00	780.00	304.00	640.00	920.00	4,454.00
9/1/2003	190.00	540.00	1,050.00	45.00	456.00	314.00	620.00	940.00	4,155.00
10/1/2003	200.00	520.00	1,100.00	20.00	345.00	324.00	600.00	960.00	4,069.00
11/1/2003	210.00	500.00	1,150.00	20.00	278.00	334.00	580.00	980.00	4,052.00
12/1/2003	220.00	480.00	1,200.00	25.00	34.00	344.00	560.00	1,000.00	3,863.00
Grand Total	$ 1,980.00	$ 7,080.00	$11,100.00	$ 1,370.00	$ 5,573.00	$ 3,468.00	$ 8,040.00	$10,680.00	$49,291.00

saxlAutoFormatTable6

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	$ 110.00	$ 700.00	$ 650.00	$ 245.00	$ 780.00	$ 234.00	$ 780.00	$ 780.00	$ 4,279.00
2/1/2003	120.00	680.00	700.00	220.00	600.00	244.00	760.00	800.00	4,124.00
3/1/2003	130.00	660.00	750.00	195.00	300.00	254.00	740.00	820.00	3,849.00
4/1/2003	140.00	640.00	800.00	170.00	900.00	264.00	720.00	840.00	4,474.00
5/1/2003	150.00	620.00	850.00	145.00	300.00	274.00	700.00	860.00	3,899.00
6/1/2003	160.00	600.00	900.00	120.00	300.00	284.00	680.00	880.00	3,924.00
7/1/2003	170.00	580.00	950.00	95.00	500.00	294.00	660.00	900.00	4,149.00
8/1/2003	180.00	560.00	1,000.00	70.00	780.00	304.00	640.00	920.00	4,454.00
9/1/2003	190.00	540.00	1,050.00	45.00	456.00	314.00	620.00	940.00	4,155.00
10/1/2003	200.00	520.00	1,100.00	20.00	345.00	324.00	600.00	960.00	4,069.00
11/1/2003	210.00	500.00	1,150.00	20.00	278.00	334.00	580.00	980.00	4,052.00
12/1/2003	220.00	480.00	1,200.00	25.00	34.00	344.00	560.00	1,000.00	3,863.00
Grand Total	$ 1,980.00	$ 7,080.00	$11,100.00	$ 1,370.00	$ 5,573.00	$ 3,468.00	$ 8,040.00	$10,680.00	$49,291.00

saxlAutoFormatTable7

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatTable8

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatTable9

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatTable10

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatTable11

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport1

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport2

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport3

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport4

Sum of Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport5

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	**1980**	**7080**	**11100**	**1370**	**5573**	**3468**	**8040**	**10680**	**49291**

saxlAutoFormatReport7

Sales Month	Product A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport8

Sales Month	Product A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport9

Sales Month	Product A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport10

Sales Month	Product A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport11

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport12

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport13

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport14

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport15

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport16

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport17

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport18

Sales	Product								
Month	A	B	C	D	E	F	G	H	G
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport19

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport20

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport21

Sales	Product								
Month	A	B	C	D	E	F	G	H	G
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport22

Sales	Product								
Month	A	B	C	D	E	F	G	H	Gr
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatReport23

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

saxlAutoFormatBlank

Sales	Product								
Month	A	B	C	D	E	F	G	H	Grand Total
1/1/2003	110	700	650	245	780	234	780	780	4279
2/1/2003	120	680	700	220	600	244	760	800	4124
3/1/2003	130	660	750	195	300	254	740	820	3849
4/1/2003	140	640	800	170	900	264	720	840	4474
5/1/2003	150	620	850	145	300	274	700	860	3899
6/1/2003	160	600	900	120	300	284	680	880	3924
7/1/2003	170	580	950	95	500	294	660	900	4149
8/1/2003	180	560	1000	70	780	304	640	920	4454
9/1/2003	190	540	1050	45	456	314	620	940	4155
10/1/2003	200	520	1100	20	345	324	600	960	4069
11/1/2003	210	500	1150	20	278	334	580	980	4052
12/1/2003	220	480	1200	25	34	344	560	1000	3863
Grand Total	1980	7080	11100	1370	5573	3468	8040	10680	49291

Installation

- Installing ExcelWriter
- Removing ExcelWriter
- Configuring IIS
- LockedFileRetry Parameter
- Security Considerations

Installing ExcelWriter

- Upgrading to a New Version of ExcelWriter
- Automatic Installation
- ExcelWriter Installation Files
- Manual Installation

Upgrading to a New Version of ExcelWriter

If you are upgrading to a new version of ExcelWriter before installing the new version, remove the old version.

Automatic Installation (OfficeWriter 4.5.1 and earlier)

> Starting in OfficeWriter 4.6.0, ExcelWriter COM is no longer distributed with the installer. These instructions are for performing an automatic installation with OfficeWriter 4.5.1 or earlier. To install later versions of ExcelWriter COM, see the manual installation instructions below.

To run the automatic installation, double-click OfficeWriter-4.x.x-Win.exe and follow the InstallShield instructions. The automatic installation will:

- Unpack ExcelWriter files to C:\Program Files\SoftArtisans\OfficeWriter\ExcelWriter or a location you selected during installation,
- Register SAXW7COM.dll and
- Create a virtual directory called OfficeWriter\ExcelWriter containing documentation and samples.

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 LicenseManager.exe (included with ExcelWriter).

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
SAXW7COM.dll	SAXW7COM.dll contains: • The COM version of ExcelTemplate. • The COM version of ExcelApplication. SAXW7COM.dll is installed in OfficeWriter\ExcelWriter\bin\dotnet10. SAXW7COM.dll must be registered with regsvr32. The automatic installation registers SAXW7COM.dll; for manual registration instructions see Manual Installation.
SoftArtisans.OfficeWriter.ExcelWriter.dll	SoftArtisans.OfficeWriter.ExcelWriter.dll is included only with the .NET version of ExcelWriter. If the ASP version of ExcelWriter was selected during automatic installation, SoftArtisans.OfficeWriter.ExcelWriter.dll will not be installed. SoftArtisans.OfficeWriter.ExcelWriter.dll contains: • The .NET version of ExcelTemplate object. • The .NET version of ExcelApplication. SoftArtisans.OfficeWriter.ExcelWriter.dll is installed in OfficeWriter\ExcelWriter\Samples\bin\dotnet20. 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.

Manual Installation

1. Obtain the ExcelWriter COM installation files:
   a. OfficeWriter 4.6.0-4.6.1 - contact SoftArtisans to obtain a copy of ExcelWriter COM
   b. OfficeWriter 4.0.0-4.5.1 - run the automatic installer to unpack the installation files
2. Open a command prompt window and move to the directory **OfficeWriter\ExcelWriter\bin\dotnet10**.
   For example, enter cd C:\Program Files\SoftArtisans\OfficeWriter\ExcelWriter\bin\dotnet10.
3. Enter regsvr32 SAXW7COM.dll. A dialog box will inform you that SAXW7COM.dll registered successfully; click Ok.
4. Enter iisreset to reset IIS.
5. Run LicenseManager.exe to add the OfficeWriter license key to the registry.

Removing ExcelWriter

Automatic Removal (OfficeWriter 4.5.1 and earlier)

> ⚠ Since ExcelWriter COM is no longer distributed with the installer starting in v4.6.0, v4.6.0 and 4.6.1 installers will not remove ExcelWriter COM. See manual removal instructions below.

If ExcelWriter was installed with the automatic installer, use **Add/Remove Programs** to unregister and 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. **Unregister SAXW7COM.dll:**
    a. Open a DOS prompt window.
    b. Go to the folder OfficeWriter\ExcelWriter.
    c. Enter regsvr32 SAXW7COM.dll /u and click Ok.
    d. To reset IIS, enter iisreset.

    > It may be necessary to stop IIS before unregistering SAXW7COM.dll. To stop IIS, at the command prompt enter:
    > net stop iisadmin

2. **Remove ExcelWriter's license key:**
    a. Double-click **LicenseManager.exe**.
    b. Select ExcelWriter's license key.
    c. Click **Remove Key**.
3. **Delete SAXW7COM.dll.**

Configuring IIS

When a spreadsheet is streamed directly to the client (rather than saved on the Web server's hard disk), an "Open or Save" dialog box will open in the browser. With Internet Explorer, it is possible to prevent the display of this dialog box, and open the spreadsheet automatically in the browser window. To do this, you will need to:

1. Associate Excel's xls extension with asp.dll when configuring your virtual directory:
    a. Open the Internet Services Manager.
    b. Right click the virtual directory containing your ExcelWriter scripts and select Properties.
    c. Click Configuration. If the Configuration button is disabled, first click Create.
    d. Select the App Mappings tab.
    e. Click Add.
    f. Browse to select asp.dll, typically in c:\windows\system32\inetsrv.
    g. Enter xls in the Extension text box.
    h. Select "All Verbs."
    i. Check "Script Engine."
    j. **Do not** check "Check that file exists."
    k. Click Ok.
2. Rename your ExcelWriter scripts with the extension .xls. For example, at the DOS command prompt, enter "ren xlgen.asp xlgen.xls."

If you associate the xls extension with asp.dll within the settings of a virtual directory, **do not** save standard Excel spreadsheets in this directory. If you do, ASP will attempt to interpret these Excel spreadsheets as ASP script and fail.

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.

LockedFileRetry Parameter

When an ExcelWriter script opens an existing spreadsheet (using ExcelApplication.Open or ExcelTemplate.Open), the spreadsheet is locked until the request is complete. To prevent failure of additional requests for the same script while the first request is processing, ExcelWriter will retry a request that could not be processed because the spreadsheet was locked. By default, ExcelWriter will retry the request every 2 seconds. To change this value:

1. Open the **Start** menu and select **Run...**
2. Enter **regedit** and click **Ok**.
3. Open the folder HKEY_LOCAL_MACHINE\SOFTWARE\SoftArtisans\ExcelWriter\Parameters.
4. Double-click the **LockedFileRetry** key. An **Edit DWORD Value** dialog will open.
5. In the **Value Data** field, enter a new value in seconds.

Security Considerations

In ASP, ExcelWriter executes in the context of the IUSR_*MACHINENAME* (the default anonymous user account) or an authenticated user account. When an ExcelWriter application opens an existing spreadsheet - either using ExcelTemplate or ExcelApplication.Open - the IUSR_*MACHINENAME* account or the authenticated user account must have **Read** access to this spreadsheet.

If you want to save a generated Excel file on the Web server (rather than stream it to the browser), the IUSR_*MACHINENAME* account or the authenticated user account must have **Write** access to the directory in which the file will be saved.

OfficeWriter Links

In addition to ExcelWriter, OfficeWriter includes the following products:

- **ExcelWriter**
  - **ExcelWriter** allows you to create Excel spreadsheets on the server programmatically or from a template.
- **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.

External Links

- OfficeWriter Online
- OfficeWriter Forums
- SoftArtisans Technical Services
- SoftArtisans
- General Questions