

Table of Contents

Office File API (Home Page)

Demo Application	30
Redistribution and Deployment	32
Spreadsheet Document API	34
Product Structure	36
Getting Started	37
Spreadsheet Document	40
Workbook	41
Worksheets.....	43
Cells and Cell Ranges.....	45
Rows and Columns.....	47
Comments.....	49
Shapes, Pictures, Charts.....	51
Measure Units.....	53
Supported Formats	54
Cell Basics	55
Cell Data Types.....	56
Dates and Times in Cells.....	60
Error Types.....	63
Cell Referencing.....	65
Formatting Cells.....	71
Formulas	74
Functions	76
Mathematical Functions.....	77
Statistical Functions.....	85
Date and Time Functions.....	94
Text Functions.....	100
Financial Functions.....	101
Logical Functions.....	104
Lookup and Reference Functions.....	105
Engineering Functions.....	107
Information Functions.....	111
Compatibility Functions.....	113
Database Functions.....	115
Web Functions.....	116
User-Defined Functions (UDF).....	117
Real Time Data (RTD) function.....	118
Operators.....	119
Array Formulas.....	121
Formula Engine.....	123
Defined Names	124
Data Binding	128
Pivot Tables	130
Pivot Table Structure.....	134
Mail Merge	136
Mail Merge Overview.....	137
Template Document.....	138
Mail Merge Functions.....	141
Examples	144
Files	148

How to: Load a Document to a Workbook.....	149
How to: Save a Document to a File.....	150
How to: Export a Workbook to PDF.....	152
How to: Export a Document to HTML.....	153
Workbooks.....	154
How to: Create a New Workbook.....	155
How to: Clone a Workbook.....	156
How to: Specify Document Properties.....	157
How to: Merge Multiple Workbooks Into One Document.....	160
Worksheets.....	161
How to: Access a Worksheet.....	162
How to: Set an Active Worksheet.....	163
How to: Add a New Worksheet.....	164
How to: Delete a Worksheet.....	166
How to: Rename a Worksheet.....	167
How to: Copy Worksheets within a Workbook.....	168
How to: Copy Worksheets between Workbooks.....	169
How to: Move a Worksheet to another Location.....	170
How to: Show and Hide a Worksheet.....	172
How to: Show and Hide Gridlines.....	174
How to: Show and Hide Row and Column Headings.....	176
How to: Set Page Orientation.....	178
How to: Set Page Margins.....	179
How to: Set Paper Size.....	180
How to: Zoom In and Out of a Worksheet.....	181
Rows and Columns.....	182
How to: Access a Row or Column.....	183
How to: Add a New Row or Column to a Worksheet.....	185
How to: Delete a Row or Column from a Worksheet.....	187
How to: Copy a Row or Column.....	189
How to: Hide a Row or a Column.....	190
How to: Specify Row Height or Column Width.....	191
Cells.....	194
How to: Access a Cell in a Worksheet.....	195
How to: Access a Range of Cells.....	197
How to: Insert a Cell or Cell Range.....	199
How to: Delete a Cell or Range of Cells.....	200
How to: Create a Named Range of Cells.....	201
How to: Change a Cell or Cell Range Value.....	203
How to: Add Formulas to Cells.....	206
How to: Add a Hyperlink to a Cell.....	207
How To: Add a Comment To a Cell.....	208
How to: Clear Cells of Content, Formatting, Hyperlinks and Comments.....	210
How to: Copy Cell Data Only, Cell Style Only, or Cell Data with Style.....	212
How to: Merge Cells or Split Merged Cells.....	214
Formulas.....	216
How to: Use Constants and Calculation Operators in Formulas.....	217
How to: Use Cell and Worksheet References in Formulas.....	218
How to: Use Names in Formulas.....	221
How to: Create Named Formulas.....	222
How to: Use Functions and Nested Functions in Formulas.....	224
How to: Create Shared Formulas.....	226
How to: Create Array Formulas.....	227
Data Import and Export.....	228
How to: Import Data to a Worksheet.....	229
How to: Export a Worksheet Range to a DataTable.....	232
Data Binding.....	235
How to: Use Worksheet Table as a Data Source.....	236

How to: Use Worksheet Data Bindings to Log and Process Data.....	241
Sorting	243
Mail Merge.....	245
How to: Perform a Mail Merge.....	246
How to: Validate the ObjectDataSource Contained in the Spreadsheet MailMerge Template.....	248
Search	250
Charts	252
How to: Create a Basic Chart.....	253
How to: Format Chart Elements	261
How to: Display the Chart Title.....	266
How to: Display and Format Data Label.....	268
How to: Show or Hide the Chart Legend.....	273
How to: Change the Display of Chart Axes.....	275
How to: Protect a Chart.....	284
Sparklines.....	285
How to: Create Sparklines.....	286
How to: Group and Ungroup Sparklines.....	288
How to: Customize the Sparkline Appearance.....	290
How to: Specify Sparkline Axis Settings.....	292
How to: Delete Sparklines.....	295
Formatting.....	297
How to: Apply a Style to a Cell or Range of Cells	298
How to: Create or Modify a Style.....	300
How to: Format a Cell or Range of Cells.....	303
How to: Specify Number or Date Format for Cell Content.....	307
How to: Change Cell Font and Background Color.....	311
How to: Configure Cell Font Settings.....	313
How to: Align Cell Content.....	315
How to: Add and Remove Cell Borders.....	317
How to: Clear Cell Formatting.....	321
Conditional Formatting.....	322
How to: Format Cell Values that are Above or Below the Average.....	323
How to: Format Cells that are Between or Not Between Two Values.....	325
How to: Format Top or Bottom Ranked Values.....	327
How to: Format Cells based on the Text in the Cell.....	329
How to: Format Unique or Duplicate Values, Blank Cells and Formula Errors.....	331
How to: Format Cells that Contain Dates.....	333
How to: Format Cells that are Less than, Greater than or Equal to a Value.....	334
How to: Use a Formula to Determine which Cells to Format.....	336
How to: Format Cells Using a Two-Color Scale.....	338
How to: Format Cells Using a Three-Color Scale.....	340
How to: Format Cells Using Data Bars.....	342
How to: Format Cells Using Icon Sets.....	346
Group Data.....	349
How to: Outline Data Manually.....	350
How to: Outline Data Automatically.....	353
How to: Insert Subtotals in a Data Range.....	354
Filter Data.....	356
How to: Enable Filtering.....	357
How to: Filter by Cell Values.....	360
How to: Filter by Date Values.....	361
How to: Apply a Custom Date Filter.....	362
How to: Apply a Custom Text Filter.....	363
How to: Apply a Custom Number Filter.....	364
How to: Apply a Dynamic Filter.....	365
How to: Filter Top or Bottom Ranked Values.....	366
How to: Sort Data in the Filtered Range.....	367
How to: Reapply a Filter.....	369

How to: Clear a Filter.....	370
Tables	372
How to: Create a Table.....	373
How to: Perform Calculations in a Table.....	374
How to: Apply a Table Style.....	378
How to: Create, Modify And Delete Table Styles.....	380
Pivot Tables.....	386
How to: Create a Pivot Table.....	387
How to: Refresh a Pivot Table.....	390
How to: Change a Data Source for a Pivot Table.....	391
How to: Move a Pivot Table.....	392
How to: Clear or Remove a Pivot Table.....	394
How to: Change the PivotTable Layout.....	397
How to: Subtotal Fields in a Pivot Table.....	401
How to: Display or Hide Grand Totals for a Pivot Table.....	404
How to: Apply a Predefined Style to a Pivot Table.....	407
How to: Apply a Custom Style to a Pivot Table.....	409
How to: Control Style Options.....	412
How to: Change the Summary Function for a Data Field.....	415
How to: Apply a Custom Calculation to a Data Field.....	417
How to: Create a Calculated Field.....	420
How to: Create a Calculated Item.....	424
How to: Sort Items in a Pivot Table.....	429
How to: Filter Items in a Pivot Table.....	431
How to: Group Items in a Pivot Table.....	441
Printing	447
How to: Print a Workbook.....	448
How to: Specify Print Settings.....	450
How to: Show a Print Preview Form for a Workbook.....	452
How to: Add Headers and Footers to a Worksheet Printout.....	453
How to: Define a Print Area.....	456
How to: Print Titles on a Worksheet.....	457
How to: Use the WPF Chart Rendering Mechanism When Printing or Exporting a Workbook to PDF	458
Pictures	459
How to: Insert and Delete Pictures.....	460
How to: Modify an Embedded Picture.....	462
Protection.....	464
How to: Protect a Workbook.....	465
How to: Protect a Worksheet.....	467
How to: Protect Specific Worksheet Ranges.....	469
Word Processing Document API	471
Getting Started	473
Word Processing Document	476
Document Structure.....	477
Document Model.....	478
Document Layout.....	481
Document Elements.....	482
Positions and Ranges.....	483
Characters.....	485
Paragraphs.....	486
Pictures	487
Hyperlinks and Bookmarks.....	490
Headers and Footers.....	491
Tables	492
Sections	493
Styles	494

Numbered and Bulleted Lists	495
Merge and Split Documents	496
Merge Documents	497
Import and Export	501
Fields	511
Field Codes.....	515
AUTHOR	517
COMMENTS.....	518
CREATEDATE.....	519
DATE	520
DOCPROPERTY.....	521
DOCVARIABLE.....	523
HYPERLINK.....	524
IF	525
INCLUDEPICTURE.....	526
KEYWORDS.....	527
LASTSAVEDBY.....	528
MERGEFIELD.....	529
NUMPAGES.....	530
PAGE	531
PRINTDATE.....	532
REVNUM	533
SAVEDATE.....	534
SEQ	535
SUBJECT	536
SYMBOL	537
TC	538
TIME	539
TITLE	540
TOC	541
Format Switches	542
General Format Switch.....	543
Numeric Format Switch.....	545
Date and Time Format Switch.....	546
Mail Merge	547
Mail Merge.....	548
Master-Detail Report.....	550
Printing	557
Export to PDF	562
Document Protection	567
Floating Objects	569
HTML Tag Support	570
Examples	573
Files	575
How to: Create a New Document.....	576
How to: Load a Document.....	577
How to: Save a Document.....	580
Document Conversion.....	582
How To: Convert a DOCX Document to PDF format.....	583
How To: Convert a DOCX Document to HTML format.....	584
How To: Convert an HMTL Document to PDF format.....	585
How To: Convert an HTML Document to DOCX format.....	586
Text	587
How to: Select Text Programmatically.....	588
How to: Insert Text at the Cursor Position.....	589
How to: Append Text to the Paragraph.....	590
How to: Copy and Paste Range.....	591
Formatting.....	592

How to: Change Formatting of Selected Text.....	593
How to: Change Formatting of the Current Paragraph.....	595
How to: Specify Default Document Formatting.....	597
Styles	598
How To: Create New Character Style.....	599
How To: Create New Paragraph Style.....	600
How To: Create New Linked Style.....	601
Lists	603
How to: Create Bulleted List in Code.....	604
How to: Create Numbered List in Code.....	606
Pictures	609
How to: Insert Inline Picture.....	610
How to: Resize Inline Picture.....	611
How to: Save Inline Picture to a File.....	612
How to: Add Floating Picture.....	613
How to: Position Floating Picture.....	614
How to: Change Z-Order and Text Wrapping.....	615
Text Boxes.....	616
How to: Insert Text Box.....	617
How to: Insert Rich Text in the Text Box.....	618
How to: Distinguish Between Text and Picture Shapes.....	619
Tables	620
How to: Insert a Table.....	621
How to: Create a Table with Fixed Column Width.....	623
How to: Merge and Split Table Cells.....	624
How to: Change Table Color.....	626
How to: Create and Apply Table Style.....	628
How to: Use Conditional Style.....	631
How to: Delete a Table.....	633
Document Elements.....	634
How to: Insert Bookmark or Hyperlink.....	635
How to: Create, Edit and Delete Comments.....	637
How to: Create and Modify Header.....	642
How to: Specify Document Properties.....	644
How to: Insert and Modify Fields.....	647
How to: Create a Checkbox.....	650
Layout	651
How to: Configure the Page Layout Programmatically.....	652
How to: Create a Three-Column Layout with Uniform Columns.....	653
How To: Add Line Numbering.....	654
Protection.....	655
How to: Protect a Document.....	656
How to: Grant Editing Permissions in a Document.....	658
Printing	659
How to: Print a Document.....	660
How to: Print a Document using the PrintableComponentLink.....	661
How to: Show a Print Preview Form.....	666
Export	668
How to: Save Selected Image to a File.....	669
How to: Export Range To Different Formats.....	670
How to: Determine Formatting Capabilities Required to Export the Document.....	672
How to: Retain the Image URI in HTML Document.....	673
PDF Document API	675
Getting Started	679
Coordinate Systems	683
Generating a Document	687
PDF Graphics	689

Overview	690
Creating PDF Graphics Context	692
Drawing into Graphics Context	693
Adding Graphics Content to a Page	695
Adding Interactive Form Fields	696
Document Manipulation	699
Merging Documents	700
Page Manipulation	701
Additional Content	706
Bookmarks	707
Hyperlinks	710
Attachments	712
Interactive Forms	714
Interactive Form Filling	715
Interactive Form Flattening	717
Creating Interactive Form	720
Deleting Interactive Form	723
Export and Import Interactive Form Data	725
Text Markup Annotations	727
Creating	728
Modifying	730
Deleting	732
Document Security	734
Protecting a Document	735
Signing a Document	738
Examples	739
Document Creation API	741
How to: Generate a Document Layout from Scratch	742
How to: Create Graphics in a Document with Landscape and Portrait Page Orientations	744
How to: Add a Link to a Page	748
How to: Add a Link to URI	750
How to: Add Bookmarks to a Document	752
How to: Bookmark Search Results in a Document	754
How to: Highlight Search Results in a Document	756
How to: Create an Interactive Form	758
How to: Attach a File to a Document	760
Interactive Form	762
How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group	763
How to: Obtain a Checked Appearance Name for a Check Box	767
How to: Fill an Interactive Form with Values	771
How to: Export Interactive Form Data to XML	773
How to: Import Interactive Form Data from XML	774
How to: Flatten Interactive Form Fields in a Document	775
How to: Add Interactive Form Fields to a PDF Document	777
How to: Delete Interactive Form Fields from a PDF Document	779
Text Markup Annotation	781
How to: Add a Text Markup Annotation to a PDF Document	782
How to: Modify an Existing Text Markup Annotation	784
How to: Delete Particular Markup Annotations from a Page	786
Extract Content from a PDF Document	788
How to: Extract Images from a Document	789
How to: Extract Text from a Document	791
How to: Search Text in a Document	792
Manage Pages of a PDF Document	794
How to: Extract Pages from a Document	795
How to: Delete Pages from a Document	797
How to: Merge Documents into a Single PDF File	798
How to: Rotate Pages	799

Document Protection.....	800
How to: Protect a PDF Document with a Password.....	801
How to: Add a Digital Signature into a PDF Document.....	803
Printing	805
How to: Use the PDF Printer Settings.....	806
How to: Customize PDF Print Output.....	808
Export a PDF Document to an Image.....	811
How to: Export a PDF Document to a Bitmap.....	811
How to: Export a PDF Document to a Multi-Page Tiff.....	813
Excel Export Library	814
Overview	815
Excel Export and Spreadsheet Document API Architecture.....	816
Excel Export and Spreadsheet Document API Feature Comparison.....	818
Excel Export Specifications and Limits	820
Product Structure	822
Getting Started	825
Examples	832
Workbooks.....	834
How to: Create a New Document.....	835
How to: Password Protect a Workbook.....	836
How to: Specify Document Properties for a Workbook.....	838
Worksheets.....	840
How to: Create a New Worksheet.....	841
How to: Set a Worksheet Name.....	842
How to: Hide a Worksheet.....	843
How to: Show and Hide Row and Column Headers.....	845
How to: Show and Hide Gridlines.....	847
Rows and Columns.....	849
How to: Create a Column.....	850
How to: Create a Row.....	851
How to: Hide a Row or Column.....	852
How to: Specify Row Height and Column Width.....	853
How to: Freeze Rows and Columns.....	855
Cells	857
How to: Create a Worksheet Cell and Set Its Value.....	858
How to: Merge Cells or Split Merged Cells.....	862
How to: Add a Hyperlink to a Cell.....	865
Formulas.....	868
How to: Create a Cell Formula.....	869
How to: Create Shared Formulas.....	874
How to: Create Subtotals.....	876
Formatting.....	878
How to: Format a Cell.....	879
How to: Apply Predefined Formatting to a Cell.....	882
How to: Apply Themed Formatting to a Cell.....	890
How to: Change Cell Background Color.....	894
How to: Configure Cell Font Settings.....	897
How to: Add Cell Borders.....	903
How to: Align Cell Content.....	906
How to: Specify Number Format for Cell Content.....	912
How to: Apply Rich Formatting to the Cell Text.....	922
Conditional Formatting.....	924
How to: Format Cell Values that are Above or Below the Average.....	925
How to: Format Cells that are Less than, Greater than or Between a Value.....	929
How to: Format Blank Cells.....	933
How to: Format Unique or Duplicate Values.....	935
How to: Use a Formula to Determine what Cells to Format.....	937

How to: Format Top or Bottom Ranked Values	939
How to: Format Cells based on the Text in the Cell	941
How to: Format Cells with Dates	943
How to: Format Cells Using Data Bars	945
How to: Format Cells Using Icon Sets	949
How to: Format Cells Using Color Scales	953
Tables	955
How to: Create a Table	956
How to: Apply a Table Style	959
How to: Apply Custom Formatting to a Table	968
How to: Create a Calculated Column	972
Grouping	975
Filtering	982
Sparklines	1004
How to: Create Sparklines	1005
How to: Customize the Sparkline Appearance	1006
How to: Specify Sparkline Axis Settings	1008
Pictures	1011
How to: Insert and Position a Picture in a Worksheet	1012
How to: Add a Hyperlink to a Picture	1016
Printing	1018
How to: Specify Print Settings	1019
How to: Set Page Margins	1021
How to: Add Headers and Footers to a Worksheet Printout	1022
How to: Insert Page Breaks in a Worksheet	1025
How to: Print Titles on a Worksheet	1026
Data Validation	1027
How to: Apply Data Validation	1028
Snap Report API	1031
Getting Started	1032
Concepts	1045
Application Programming Interface	1046
Snap Fields	1048
File Formats	1054
Examples	1055
How to: Insert Data	1056
How to: Format Data	1058
How to: Group Data	1062
How to: Filter Data	1065
How to: Sort Data	1066
How to: Create Parameter	1067
How to: Create Calculated Field	1069
How to: Create Barcode	1070
How to: Create Sparkline	1071
How to: Create Check Box	1072
How to: Generate Master-Detail Mail Merge Documents	1073
Zip Compression and Archive API	1074
Getting Started	1075
Examples	1076
How to: Archive a Directory	1077
How to: Archive Selected Files	1078
How to: Add File to File Archive	1079
How to: Filter Files to Archive	1080
How to: Cancel Archiving	1082
How to: Password Protect Archive Files	1083
How to: Add Comment to File	1084

How to: Compress .NET Stream.....	1085
How to: Compress Byte Array.....	1086
How to: Unzip an Archive.....	1087
How to: Resolve File Conflicts when Unzipping.....	1088
Barcode Generation API	1090
Getting Started	1091
Concepts	1092
Bar Code Types.....	1093
Codabar.....	1094
Code 11 (USD-8).....	1095
Code 128.....	1096
Code 39 (USD-3).....	1097
Code 39 Extended.....	1098
Code 93.....	1099
Code 93 Extended.....	1100
EAN 13	1101
EAN 8	1102
Data Matrix (ECC200).....	1103
GS1- Data Matrix.....	1104
EAN-128 (UCC).....	1105
Industrial 2 of 5.....	1106
Intelligent Mail.....	1107
Interleaved 2 of 5.....	1108
MSI - Plessey.....	1109
Matrix 2 of 5.....	1110
PDF417	1111
PostNet	1112
QR Code.....	1113
UPC Supplemental 2.....	1114
UPC Supplemental 5.....	1115
UPC-A	1116
UPC-E0	1117
UPC-E1	1118
GS1 DataBar.....	1119
UPC Shipping Container Symbol (ITF-14).....	1120
Bar Code Options	1121
Bar Code Recognition Specifics.....	1122
Unit Conversion API	1124
Getting Started	1127
Examples	1129
How to: Convert Between Metric Prefixes.....	1130
How to: Convert Between Metric and US Units of Measurement.....	1131
API Reference	1133
DevExpress.BarCodes	1133
BarCode Class.....	1136
BarCode Members.....	1137
BarCode Constructor.....	1140
BarCode Properties.....	1141
AutoSize Property.....	1144
BackColor Property.....	1145
BarCodeImage Property.....	1146
BarHeight Property.....	1147
BorderColor Property.....	1148
BorderWidth Property.....	1149
BottomCaption Property.....	1150

Code Property.....	1151
CodeBinaryData Property.....	1152
CodeText Property.....	1153
CodeTextFont Property.....	1154
CodeTextHorizontalAlignment Property.....	1155
CodeTextVerticalAlignment Property.....	1156
Dpi Property.....	1157
DpiX Property.....	1158
DpiY Property.....	1159
ForeColor Property.....	1160
ImageHeight Property.....	1161
ImageSize Property.....	1162
ImageWidth Property.....	1163
IsPrintingAvailable Property.....	1164
Margins Property.....	1165
Module Property.....	1166
Options Property.....	1167
Paddings Property.....	1168
RotationAngle Property.....	1169
ShowText Property.....	1170
Symbology Property.....	1171
TextRenderingHint Property.....	1172
TopCaption Property.....	1173
Unit Property.....	1174
BarCode Methods.....	1175
Dispose Method.....	1176
ExportToPdf Method.....	1177
ExportToPdf Method (Stream).....	1177
ExportToPdf Method (string).....	1177
Print Method.....	1179
PrintDialog Method.....	1180
Save Method.....	1181
Save Method (string, ImageFormat).....	1181
Save Method (Stream, ImageFormat).....	1181
ShowPrintPreview Method.....	1183
BarCodeCaption Class.....	1184
BarCodeCaption Members.....	1186
BarCodeCaption Properties.....	1187
Font Property.....	1188
ForeColor Property.....	1189
HorizontalAlignment Property.....	1190
Paddings Property.....	1191
Text Property.....	1192
BarCodeGeneratorOptions Class.....	1193
BarCodeGeneratorOptions Members.....	1194
BarCodeGeneratorOptions Properties.....	1195
ShowCodeText Property.....	1196
BarCodeMargins Class.....	1197
BarCodeMargins Members.....	1198
BarCodeMargins Constructor.....	1200
BarCodeMargins Properties.....	1201
Bottom Property.....	1203
Left Property.....	1204
Right Property.....	1205
Top Property.....	1206
BarCodeOptions Class.....	1207
BarCodeOptions Members.....	1208
BarCodeOptions Constructor.....	1211

BarCodeOptions Properties.....	1212
Codabar Property.....	1215
Code11 Property.....	1216
Code128 Property.....	1217
Code39 Property.....	1218
Code39Extended Property.....	1219
Code93 Property.....	1220
Code93Extended Property.....	1221
CodeMSI Property.....	1222
DataBar Property.....	1223
DataMatrix Property.....	1224
DataMatrixGS1 Property.....	1225
EAN128 Property.....	1226
EAN13 Property.....	1227
EAN8 Property.....	1228
Industrial2of5 Property.....	1229
IntelligentMail Property.....	1230
Interleaved2of5 Property.....	1231
ITF14 Property.....	1232
Matrix2of5 Property.....	1233
PDF417 Property.....	1234
PostNet Property.....	1235
QRCode Property.....	1236
UPCA Property.....	1237
UPCE0 Property.....	1238
UPCE1 Property.....	1239
UPCSupplemental2 Property.....	1240
UPCSupplemental5 Property.....	1241
BarCodePadding Class.....	1242
BarCodePadding Members.....	1243
BarCodePadding Constructor.....	1245
BarCodePadding Properties.....	1246
Bottom Property.....	1248
Left Property.....	1249
Right Property.....	1250
Top Property.....	1251
CodabarOptions Class.....	1252
CodabarOptions Members.....	1253
CodabarOptions Constructor.....	1255
CodabarOptions Properties.....	1256
StartStopPair Property.....	1258
WideNarrowRatio Property.....	1259
CodabarStartStopPair Enumeration.....	1260
Code11Options Class.....	1261
Code11Options Members.....	1262
Code11Options Constructor.....	1263
Code128CharacterSet Enumeration.....	1264
Code128Options Class.....	1265
Code128Options Members.....	1266
Code128Options Constructor.....	1267
Code128Options Properties.....	1268
Charset Property.....	1269
Code39ExtendedOptions Class.....	1270
Code39ExtendedOptions Members.....	1271
Code39ExtendedOptions Constructor.....	1273
Code39Options Class.....	1274
Code39Options Members.....	1275
Code39Options Constructor.....	1277

Code39Options Properties.....	1278
CalcChecksum Property.....	1280
WideNarrowRatio Property.....	1281
Code93ExtendedOptions Class.....	1282
Code93ExtendedOptions Members.....	1283
Code93ExtendedOptions Constructor.....	1284
Code93Options Class.....	1285
Code93Options Members.....	1286
Code93Options Constructor.....	1287
Code93Options Properties.....	1288
CalcChecksum Property.....	1289
CodeMSIOptions Class.....	1290
CodeMSIOptions Members.....	1291
CodeMSIOptions Constructor.....	1292
DataBarOptions Class.....	1293
DataBarOptions Members.....	1294
DataBarOptions Constructor.....	1296
DataBarOptions Properties.....	1297
FNC1Substitute Property.....	1299
SegmentsInRow Property.....	1300
Type Property.....	1301
DataBarType Enumeration.....	1302
DataMatrixCompactionMode Enumeration.....	1304
DataMatrixGS1Options Class.....	1305
DataMatrixGS1Options Members.....	1306
DataMatrixGS1Options Constructor.....	1308
DataMatrixGS1Options Properties.....	1309
FNC1Substitut Property.....	1311
FNC1Substitute Property.....	1312
DataMatrixOptions Class.....	1313
DataMatrixOptions Members.....	1314
DataMatrixOptions Constructor.....	1316
DataMatrixOptions Properties.....	1317
CompactionMode Property.....	1319
MatrixSize Property.....	1320
DataMatrixSize Enumeration.....	1321
EAN128Options Class.....	1323
EAN128Options Members.....	1324
EAN128Options Constructor.....	1326
EAN128Options Properties.....	1327
Charset Property.....	1329
FNC1Substitut Property.....	1330
FNC1Substitute Property.....	1331
EAN13Options Class.....	1332
EAN13Options Members.....	1333
EAN13Options Constructor.....	1334
EAN8Options Class.....	1335
EAN8Options Members.....	1336
EAN8Options Constructor.....	1337
Industrial2of5Options Class.....	1338
Industrial2of5Options Members.....	1339
Industrial2of5Options Constructor.....	1341
Industrial2of5Options Properties.....	1342
CalcChecksum Property.....	1344
WideNarrowRatio Property.....	1345
IntelligentMailOptions Class.....	1346
IntelligentMailOptions Members.....	1347
IntelligentMailOptions Constructor.....	1348

Interleaved2of5Options Class.....	1349
Interleaved2of5Options Members.....	1350
Interleaved2of5Options Constructor.....	1352
Interleaved2of5Options Properties.....	1353
CalcChecksum Property.....	1355
WideNarrowRatio Property.....	1356
Matrix2of5Options Class.....	1357
Matrix2of5Options Members.....	1358
Matrix2of5Options Constructor.....	1360
PDF417CompactionMode Enumeration.....	1361
PDF417ErrorLevel Enumeration.....	1362
PDF417Options Class.....	1363
PDF417Options Members.....	1364
PDF417Options Constructor.....	1366
PDF417Options Properties.....	1367
Columns Property.....	1369
CompactionMode Property.....	1370
ErrorLevel Property.....	1371
Rows Property.....	1372
TruncateSymbol Property.....	1373
PostNetOptions Class.....	1374
PostNetOptions Members.....	1375
PostNetOptions Constructor.....	1376
QRCodeCompactionMode Enumeration.....	1377
QRCodeErrorLevel Enumeration.....	1378
QRCodeOptions Class.....	1379
QRCodeOptions Members.....	1380
QRCodeOptions Constructor.....	1382
QRCodeOptions Properties.....	1383
CompactionMode Property.....	1385
ErrorLevel Property.....	1386
Version Property.....	1387
QRCodeVersion Enumeration.....	1388
Symbology Enumeration.....	1390
UPCAOptions Class.....	1397
UPCAOptions Members.....	1398
UPCAOptions Constructor.....	1399
UPCE0Options Class.....	1400
UPCE0Options Members.....	1401
UPCE0Options Constructor.....	1402
UPCE1Options Class.....	1403
UPCE1Options Members.....	1404
UPCE1Options Constructor.....	1405
UPCSupplemental2Options Class.....	1406
UPCSupplemental2Options Members.....	1407
UPCSupplemental2Options Constructor.....	1408
UPCSupplemental5Options Class.....	1409
UPCSupplemental5Options Members.....	1410
UPCSupplemental5Options Constructor.....	1411
DevExpress.Compression	1412
AllowFileOverwriteEventArgs Class.....	1414
AllowFileOverwriteEventArgs Members.....	1415
AllowFileOverwriteEventArgs Constructor.....	1417
AllowFileOverwriteEventArgs Properties.....	1418
TargetFilePath Property.....	1420
ZipItem Property.....	1421
AllowFileOverwriteEventHandler Delegate.....	1422
AllowFileOverwriteMode Enumeration.....	1423

CanContinueEventArgs Class.....	1424
CanContinueEventArgs Members.....	1425
CanContinueEventArgs Properties.....	1426
CanContinue Property.....	1427
EncryptionType Enumeration.....	1428
ErrorEventArgs Class.....	1429
ErrorEventArgs Members.....	1430
ErrorEventArgs Constructor.....	1432
ErrorEventArgs Properties.....	1433
ItemName Property.....	1435
ErrorEventArgs Methods.....	1436
GetException Method.....	1437
ErrorEventHandler Delegate.....	1438
ProgressEventArgs Class.....	1439
ProgressEventArgs Members.....	1440
ProgressEventArgs Constructor.....	1442
ProgressEventArgs Properties.....	1443
Progress Property.....	1445
ProgressEventHandler Delegate.....	1446
WrongPasswordException Class.....	1447
WrongPasswordException Members.....	1448
WrongPasswordException Constructor.....	1450
WrongPasswordException Properties.....	1451
Password Property.....	1453
ZipArchive Class.....	1454
ZipArchive Members.....	1455
ZipArchive Constructor.....	1458
ZipArchive Constructor.....	1459
ZipArchive Constructor (bool).....	1460
ZipArchive Properties.....	1461
CatchExceptions Property.....	1464
Count Property.....	1465
EncryptionType Property.....	1466
FileName Property.....	1467
Item Property.....	1468
Item Property [string].....	1468
Item Property [int].....	1468
OptionsBehavior Property.....	1470
Password Property.....	1471
ZipArchive Events.....	1472
AllowFileOverwrite Event.....	1473
Error Event.....	1475
ItemAdding Event.....	1477
Progress Event.....	1479
ZipArchive Methods.....	1481
AddByteArray Method.....	1483
AddDirectory Method.....	1485
AddDirectory Method (string).....	1485
AddDirectory Method (string, string).....	1485
AddFile Method.....	1487
AddFile Method (string).....	1487
AddFile Method (string, string).....	1488
AddFiles Method.....	1490
AddFiles Method (IEnumerable`1).....	1490
AddFiles Method (IEnumerable`1, string).....	1491
AddStream Method.....	1492
AddText Method.....	1494
AddText Method (string).....	1494

AddText Method (string, string).....	1495
AddText Method (string, string, Encoding).....	1496
Dispose Method.....	1497
Extract Method.....	1498
Extract Method.....	1498
Extract Method (string).....	1499
Extract Method (string, AllowFileOverwriteMode).....	1500
IsValid Method.....	1502
Read Method.....	1503
Read Method (string).....	1503
Read Method (Stream).....	1504
Read Method (string, Encoding).....	1505
Read Method (Stream, Encoding).....	1506
Read Method (string, Encoding, bool).....	1507
Read Method (Stream, Encoding, bool).....	1508
RemoveItem Method.....	1509
RemoveItem Method (ZipItem).....	1509
RemoveItem Method (string).....	1509
Save Method.....	1511
Save Method (Stream).....	1511
Save Method (string).....	1511
UpdateDirectory Method.....	1513
UpdateDirectory Method (string).....	1513
UpdateDirectory Method (string, string).....	1513
UpdateFile Method.....	1515
UpdateFile Method (string).....	1515
UpdateFile Method (string, string).....	1515
UpdateStream Method.....	1517
UpdateText Method.....	1518
UpdateText Method (string, string).....	1518
UpdateText Method (string, string, Encoding).....	1518
Validate Method.....	1520
ZipArchiveOptionsBehavior Class.....	1521
ZipArchiveOptionsBehavior Members.....	1522
ZipArchiveOptionsBehavior Constructor.....	1524
ZipArchiveOptionsBehavior Properties.....	1525
AllowFileOverwrite Property.....	1527
ZipByteArrayItem Class.....	1528
ZipByteArrayItem Members.....	1529
ZipByteArrayItem Constructor.....	1531
ZipByteArrayItem Properties.....	1532
Content Property.....	1534
ZipDirectoryItem Class.....	1535
ZipDirectoryItem Members.....	1536
ZipDirectoryItem Constructor.....	1538
ZipDirectoryItem Properties.....	1539
DirectoryName Property.....	1541
ZipFileItem Class.....	1542
ZipFileItem Members.....	1543
ZipFileItem Constructor.....	1545
ZipFileItem Properties.....	1546
FileName Property.....	1548
ZipItem Class.....	1549
ZipItem Members.....	1550
ZipItem Properties.....	1552
Attributes Property.....	1554
Comment Property.....	1555
CompressedSize Property.....	1556

CreationTime Property.....	1557
CreationTimeUtc Property.....	1558
Encoding Property.....	1559
EncryptionType Property.....	1560
LastAccessTime Property.....	1561
LastAccessTimeUtc Property.....	1562
LastWriteTime Property.....	1563
LastWriteTimeUtc Property.....	1564
Name Property.....	1565
Password Property.....	1566
UncompressedSize Property.....	1567
ZipItem Methods.....	1568
Extract Method.....	1569
Extract Method.....	1569
Extract Method (Stream).....	1569
Extract Method (string).....	1570
Extract Method (string, AllowFileOverwriteMode).....	1570
Open Method.....	1572
ZipItemAddingAction Enumeration.....	1573
ZipItemAddingEventArgs Class.....	1574
ZipItemAddingEventArgs Members.....	1575
ZipItemAddingEventArgs Constructor.....	1577
ZipItemAddingEventArgs Properties.....	1578
Action Property.....	1580
Item Property.....	1581
ZipItemAddingEventHandler Delegate.....	1582
ZipStreamItem Class.....	1583
ZipStreamItem Members.....	1584
ZipStreamItem Constructor.....	1586
ZipStreamItem Properties.....	1587
ContentStream Property.....	1589
ZipTextItem Class.....	1590
ZipTextItem Members.....	1591
ZipTextItem Constructor.....	1593
ZipTextItem Properties.....	1594
Content Property.....	1596
ContentEncoding Property.....	1597
DevExpress.Docs.Text	1598
EncodingDetector Class.....	1599
EncodingDetector Members.....	1600
EncodingDetector Constructor.....	1601
EncodingDetector Methods.....	1602
AnalyseData Method.....	1603
BeginDetection Method.....	1604
Detect Method.....	1605
Detect Method (Byte[]).....	1605
Detect Method (Stream).....	1605
Detect Method (Stream, int).....	1606
Detect Method (Byte[], int, int).....	1606
Detect Method (Stream, int, bool).....	1607
EndDetection Method.....	1609
INumberInWordsProvider Interface	1610
INumberInWordsProvider Members.....	1611
INumberInWordsProvider Methods.....	1612
ConvertToText Method.....	1613
ConvertToText Method (Int64).....	1613
ConvertToText Method (Int64, CultureInfo).....	1613
ConvertToText Method (Int64, NumberCulture).....	1614

NumberCulture Enumeration.....	1615
NumberInWords Class.....	1616
NumberInWords Members.....	1617
NumberInWords Properties.....	1618
Cardinal Property.....	1619
Ordinal Property.....	1620
DevExpress.Pdf	1621
PdfDocumentProcessor Class.....	1622
PdfDocumentProcessor Members.....	1625
PdfDocumentProcessor Constructor.....	1629
PdfDocumentProcessor Properties.....	1631
Document Property.....	1635
ImageCacheSize Property.....	1636
MaxPrintingDpi Property.....	1637
PasswordAttemptsLimit Property.....	1638
RenderPageContentWithDirectX Property.....	1639
ShowImagePlaceholder Property.....	1640
Text Property.....	1641
PdfDocumentProcessor Events.....	1642
PasswordRequested Event.....	1643
PrintPage Event.....	1645
QueryPageSettings Event.....	1648
PdfDocumentProcessor Methods.....	1651
AddFormFields Method.....	1654
AddFormFields Method (PdfAcroFormField[]).	1654
AddFormFields Method (IEnumerable`1).....	1656
AddNewPage Method.....	1657
AddTextMarkupAnnotation Method.....	1660
AddTextMarkupAnnotation Method (PdfDocumentArea, PdfTextMarkupAnnotationType).....	1660
AddTextMarkupAnnotation Method (int, IEnumerable`1, PdfTextMarkupAnnotationType).....	1662
AddTextMarkupAnnotation Method (int, PdfRectangle, PdfTextMarkupAnnotationType).....	1663
AddTextMarkupAnnotation Method (PdfDocumentPosition, PdfDocumentPosition, PdfTextMarkupAnnotationType).....	1665
AddTextMarkupAnnotation Method (int, PdfOrientedRectangle, PdfTextMarkupAnnotationType).....	1667
AddTextMarkupAnnotation Method (int, IEnumerable`1, PdfTextMarkupAnnotationType).....	1667
AppendDocument Method.....	1669
AppendDocument Method (Stream).....	1669
AppendDocument Method (string).....	1669
ApplyFormData Method.....	1671
AttachFile Method.....	1674
CheckFormFieldNameCollisions Method.....	1676
CheckFormFieldNameCollisions Method (PdfAcroFormField[]).	1676
CheckFormFieldNameCollisions Method (IEnumerable`1).....	1676
CloseDocument Method.....	1678
CreateBitmap Method.....	1679
CreateDestination Method.....	1681
CreateDestination Method (int).....	1681
CreateDestination Method (int, Single).....	1682
CreateDestination Method (int, Single, Single).....	1683
CreateDestination Method (int, Single, Single, Single).....	1684
CreateDestination Method (int, Single, Single, Single, Single).....	1686
CreateDestination Method (int, Single, Single, Single, Single, Single).....	1687
CreateEmptyDocument Method.....	1690
CreateEmptyDocument Method.....	1690
CreateEmptyDocument Method (string).....	1692
CreateEmptyDocument Method (Stream).....	1694
CreateEmptyDocument Method (Stream, PdfSaveOptions).....	1695
CreateEmptyDocument Method (string, PdfSaveOptions).....	1697

CreateEmptyDocument Method (Stream, PdfCreationOptions).....	1700
CreateEmptyDocument Method (string, PdfCreationOptions).....	1703
CreateEmptyDocument Method (Stream, PdfSaveOptions, PdfCreationOptions).....	1706
CreateEmptyDocument Method (string, PdfSaveOptions, PdfCreationOptions).....	1709
CreateGraphics Method.....	1713
CreateTiff Method.....	1716
CreateTiff Method (Stream, int).....	1716
CreateTiff Method (string, int).....	1717
CreateTiff Method (Stream, int, IEnumerable`1).....	1718
CreateTiff Method (string, int, IEnumerable`1).....	1720
DeleteAttachment Method.....	1722
DeleteMarkupAnnotation Method.....	1723
DeleteMarkupAnnotations Method.....	1724
DeletePage Method.....	1726
Export Method.....	1728
Export Method (string, PdfForm DataFormat).....	1728
Export Method (Stream, PdfForm DataFormat).....	1728
FindText Method.....	1730
FindText Method (string).....	1730
FindText Method (string, PdfTextSearchParameters).....	1730
FlattenForm Method.....	1732
FlattenFormField Method.....	1733
GetForm Data Method.....	1735
GetFormFieldNames Method.....	1738
GetImages Method.....	1739
GetImages Method (PdfDocumentArea).....	1739
GetImages Method (PdfDocumentPosition, PdfDocumentPosition).....	1741
GetImages Method (PdfDocumentArea, Single).....	1742
GetImages Method (PdfDocumentPosition, PdfDocumentPosition, Single).....	1742
GetMarkupAnnotationData Method.....	1744
GetPageText Method.....	1746
GetText Method.....	1747
GetText Method (PdfDocumentArea).....	1747
GetText Method (PdfDocumentPosition, PdfDocumentPosition).....	1747
Import Method.....	1749
Import Method (Stream).....	1749
Import Method (string).....	1749
Import Method (string, PdfForm DataFormat).....	1750
Import Method (Stream, PdfForm DataFormat).....	1750
InsertNewPage Method.....	1751
LoadDocument Method.....	1752
LoadDocument Method (string).....	1752
LoadDocument Method (Stream).....	1754
LoadDocument Method (string, bool).....	1756
LoadDocument Method (Stream, bool).....	1758
NextWord Method.....	1761
PrevWord Method.....	1762
Print Method.....	1763
Print Method (PdfPrinterSettings).....	1763
Print Method (PrinterSettings).....	1764
RemoveForm Method.....	1766
RemoveFormField Method.....	1768
RenderNewPage Method.....	1770
RenderNewPage Method (PdfRectangle, PdfGraphics).....	1770
RenderNewPage Method (PdfRectangle, PdfGraphics, Single, Single).....	1772
ResetForm Data Method.....	1776
SaveDocument Method.....	1777
SaveDocument Method (string).....	1777

SaveDocument Method (Stream).....	1779
SaveDocument Method (string, bool).....	1781
SaveDocument Method (Stream, bool).....	1782
SaveDocument Method (Stream, PdfSaveOptions).....	1782
SaveDocument Method (string, PdfSaveOptions).....	1784
SaveDocument Method (string, PdfSaveOptions, bool).....	1784
SaveDocument Method (Stream, PdfSaveOptions, bool).....	1785
PdfPageWord Class.....	1787
PdfPageWord Members.....	1788
PdfPageWord Properties.....	1789
PageNumber Property.....	1790
PdfViewerExtensions Class.....	1791
PdfViewerExtensions Members.....	1792
PdfViewerExtensions Methods.....	1793
Export Method.....	1794
Export Method (IPdfViewer, Stream, PdfForm DataFormat).....	1794
Export Method (IPdfViewer, string, PdfForm DataFormat).....	1795
Import Method.....	1798
Import Method (IPdfViewer, Stream).....	1798
Import Method (IPdfViewer, string).....	1799
Import Method (IPdfViewer, Stream, PdfForm DataFormat).....	1801
Import Method (IPdfViewer, string, PdfForm DataFormat).....	1803
SaveDocument Method.....	1805
SaveDocument Method (IPdfViewer, Stream, PdfSaveOptions).....	1805
SaveDocument Method (IPdfViewer, string, PdfSaveOptions).....	1806
DevExpress.Snap	1807
SnapDocumentServer Class.....	1808
SnapDocumentServer Members.....	1809
SnapDocumentServer Constructor.....	1816
SnapDocumentServer Properties.....	1817
Document Property.....	1824
Options Property.....	1825
SnxBytes Property.....	1826
SnapDocumentServer Events.....	1827
AfterDataSourceImport Event.....	1830
AsynchronousOperationFinished Event.....	1831
AsynchronousOperationStarted Event.....	1832
BeforeConversion Event.....	1833
BeforeDataSourceExport Event.....	1834
BeforeLoadCustomAssembly Event.....	1835
CustomAssemblyLoading Event.....	1836
DataSourceChanged Event.....	1837
DocumentClosing Event.....	1838
DocumentLoaded Event.....	1839
EmptyDocumentCreated Event.....	1840
MailMergeFinished Event.....	1841
MailMergeRecordFinished Event.....	1842
MailMergeRecordStarted Event.....	1843
MailMergeStarted Event.....	1844
SnapMailMergeFinished Event.....	1845
SnapMailMergeRecordFinished Event.....	1846
SnapMailMergeRecordStarted Event.....	1847
SnapMailMergeStarted Event.....	1848
ValidateCustomSql Event.....	1849
SnapDocumentServer Methods.....	1850
CreateMailMergeOptions Method.....	1853
CreateSnapMailMergeExportOptions Method.....	1854
ExportDocument Method.....	1855

ExportDocument Method (string, DocumentFormat).....	1855
ExportDocument Method (Stream, DocumentFormat).....	1855
MailMerge Method.....	1857
MailMerge Method (Document).....	1857
MailMerge Method (IRichEditDocumentServer).....	1858
MailMerge Method (MailMergeOptions, Document).....	1858
MailMerge Method (MailMergeOptions, IRichEditDocumentServer).....	1859
MailMerge Method (Stream, DocumentFormat).....	1859
MailMerge Method (string, DocumentFormat).....	1860
MailMerge Method (MailMergeOptions, Stream, DocumentFormat).....	1860
MailMerge Method (MailMergeOptions, string, DocumentFormat).....	1861
SaveDocument Method.....	1862
SaveDocument Method (string).....	1862
SaveDocument Method (Stream).....	1862
SaveDocument Method (string, DocumentFormat).....	1863
SaveDocument Method (Stream, DocumentFormat).....	1863
SnapMailMerge Method.....	1865
SnapMailMerge Method (ISnapDocumentServer).....	1865
SnapMailMerge Method (SnapDocument).....	1866
SnapMailMerge Method (Stream, DocumentFormat).....	1867
SnapMailMerge Method (SnapMailMergeExportOptions, ISnapDocumentServer).....	1868
SnapMailMerge Method (SnapMailMergeExportOptions, SnapDocument).....	1869
SnapMailMerge Method (string, DocumentFormat).....	1869
SnapMailMerge Method (SnapMailMergeExportOptions, string, DocumentFormat).....	1870
SnapMailMerge Method (SnapMailMergeExportOptions, Stream, DocumentFormat).....	1871
DevExpress.Spreadsheet	1872
Workbook Class.....	1873
Workbook Members.....	1874
Workbook Constructor.....	1887
Workbook Properties.....	1888
ChartSheets Property.....	1901
Clipboard Property.....	1902
CurrentAuthor Property.....	1903
CustomFunctions Property.....	1904
CustomXmlParts Property.....	1905
DefinedNames Property.....	1906
DocumentProperties Property.....	1908
DocumentSettings Property.....	1909
ExternalWorkbooks Property.....	1910
FormulaEngine Property.....	1912
Functions Property.....	1913
GlobalCustomFunctions Property.....	1914
HasMacros Property.....	1915
History Property.....	1916
IsDisposed Property.....	1917
IsProtected Property.....	1918
IsUpdateLocked Property.....	1919
MailMergeDataMember Property.....	1920
MailMergeDataSource Property.....	1921
MailMergeOptions Property.....	1922
MailMergeParameters Property.....	1923
Model Property.....	1924
Modified Property.....	1925
Options Property.....	1926
Path Property.....	1927
PivotCaches Property.....	1928
Range Property.....	1929
RealTimeData Property.....	1931

Sheets Property.....	1932
Styles Property.....	1933
TableStyles Property.....	1935
Tag Property.....	1936
Unit Property.....	1937
Worksheets Property.....	1938
Workbook Events.....	1939
ActiveSheetChanged Event.....	1945
ActiveSheetChanging Event.....	1946
BeforeExport Event.....	1947
BeforeImport Event.....	1948
BeforePrintSheet Event.....	1949
ClipboardDataObtained Event.....	1950
ClipboardDataPasted Event.....	1951
ClipboardDataPasting Event.....	1952
ColumnsInserted Event.....	1953
ColumnsInserting Event.....	1954
ColumnsRemoved Event.....	1955
ColumnsRemoving Event.....	1956
CommentInserted Event.....	1957
CommentInserting Event.....	1958
CommentRemoved Event.....	1959
CommentRemoving Event.....	1960
ContentChanged Event.....	1961
CopiedRangePasted Event.....	1962
CopiedRangePasting Event.....	1963
CustomAssemblyLoading Event.....	1964
DefinedNameConflictResolving Event.....	1965
DocumentClosing Event.....	1966
DocumentLoaded Event.....	1967
DocumentSaved Event.....	1968
EmptyDocumentCreated Event.....	1969
EncryptedFileIntegrityCheckFailed Event.....	1970
EncryptedFilePasswordRequest Event.....	1971
InitializeDocument Event.....	1972
InvalidFormatException Event.....	1973
ModifiedChanged Event.....	1974
PanesFrozen Event.....	1975
PanesUnfrozen Event.....	1976
RangeCopied Event.....	1977
RangeCopying Event.....	1978
RowsInserted Event.....	1979
RowsInserting Event.....	1980
RowsRemoved Event.....	1981
RowsRemoving Event.....	1982
SchemaChanged Event.....	1983
ScrollPositionChanged Event.....	1984
SelectionChanged Event.....	1985
ShapeInserted Event.....	1986
ShapeRemoved Event.....	1987
ShapeRemoving Event.....	1988
ShapesCopying Event.....	1989
SheetInserted Event.....	1990
SheetRemoved Event.....	1991
SheetRemoving Event.....	1992
SheetRenamed Event.....	1993
SheetRenaming Event.....	1994
UnitChanged Event.....	1995

UnitChanging Event.....	1996
ValidateCustomSqlQuery Event.....	1997
Workbook Methods.....	1998
AddService Method.....	2001
AddService Method (Type, object).....	2001
AddService Method (Type, ServiceCreatorCallback).....	2002
AddService Method (Type, object, bool).....	2002
AddService Method (Type, ServiceCreatorCallback, bool).....	2003
BeginUpdate Method.....	2005
Calculate Method.....	2006
Calculate Method.....	2006
Calculate Method (Worksheet).....	2006
Calculate Method (Range).....	2007
CalculateFull Method.....	2008
CalculateFullRebuild Method.....	2009
CancelUpdate Method.....	2010
CreateNewDocument Method.....	2011
Dispose Method.....	2012
EndUpdate Method.....	2013
Evaluate Method.....	2014
Evaluate Method (string).....	2014
Evaluate Method (string, FormulaEvaluationContext).....	2014
ExportToHtml Method.....	2016
ExportToHtml Method (string, Range).....	2017
ExportToHtml Method (string, int).....	2017
ExportToHtml Method (string, Worksheet).....	2018
ExportToHtml Method (string, HtmlIDocumentExporterOptions).....	2018
ExportToHtml Method (Stream, int).....	2019
ExportToHtml Method (Stream, HtmlIDocumentExporterOptions).....	2019
ExportToHtml Method (Stream, Worksheet).....	2019
ExportToHtml Method (Stream, Range).....	2020
ExportToPdf Method.....	2021
ExportToPdf Method (string).....	2021
ExportToPdf Method (Stream).....	2022
ExportToPdf Method (string, PdfExportOptions).....	2022
ExportToPdf Method (Stream, PdfExportOptions).....	2023
GenerateMailMergeDocuments Method.....	2024
GetService Method.....	2025
GetService<T> Method.....	2026
LoadDocument Method.....	2027
LoadDocument Method (string).....	2027
LoadDocument Method (Byte[]).....	2028
LoadDocument Method (Stream).....	2028
LoadDocument Method (Byte[], DocumentFormat).....	2029
LoadDocument Method (string, DocumentFormat).....	2030
LoadDocument Method (Stream, DocumentFormat).....	2031
Print Method.....	2032
Print Method.....	2032
Print Method (PrinterSettings).....	2033
Protect Method.....	2035
RemoveService Method.....	2037
RemoveService Method (Type).....	2037
RemoveService Method (Type, bool).....	2038
ReplaceService<T> Method.....	2039
SaveDocument Method.....	2040
SaveDocument Method (string).....	2040
SaveDocument Method (DocumentFormat).....	2041
SaveDocument Method (string, DocumentFormat).....	2041

SaveDocument Method (Stream, DocumentFormat).....	2042
Search Method.....	2044
Search Method (string).....	2044
Search Method (string, SearchOptions).....	2045
Unprotect Method.....	2046
WorkbookExtensions Class.....	2047
WorkbookExtensions Members.....	2048
WorkbookExtensions Methods.....	2049
Clone Method.....	2050
Clone Method (IWorkbook).....	2050
Clone Method (IWorkbook, bool).....	2051
Merge Method.....	2053
WorksheetExtensions Class.....	2055
WorksheetExtensions Members.....	2056
WorksheetExtensions Methods.....	2057
CreateDataTable Method.....	2058
CreateDataTable Method (Worksheet, Range, bool).....	2058
CreateDataTable Method (Worksheet, Range, bool, bool).....	2061
CreateDataTableExporter Method.....	2063
Import Method.....	2064
Import Method (Worksheet, String[,], int, int).....	2067
Import Method (Worksheet, Int16[,], int, int).....	2068
Import Method (Worksheet, object, int, int).....	2069
Import Method (Worksheet, Int32[,], int, int).....	2069
Import Method (Worksheet, Byte[,], int, int).....	2070
Import Method (Worksheet, Int64[,], int, int).....	2071
Import Method (Worksheet, Object[,], int, int).....	2071
Import Method (Worksheet, Single[,], int, int).....	2073
Import Method (Worksheet, Double[,], int, int).....	2073
Import Method (Worksheet, Decimal[,], int, int).....	2074
Import Method (Worksheet, DateTime[,], int, int).....	2074
Import Method (Worksheet, Boolean[,], int, int).....	2075
Import Method (Worksheet, Byte[], int, int, bool).....	2076
Import Method (Worksheet, DataTable, bool, int, int).....	2076
Import Method (Worksheet, Boolean[], int, int, bool).....	2078
Import Method (Worksheet, IDataReader, bool, int, int).....	2079
Import Method (Worksheet, IEnumerable, int, int, bool).....	2080
Import Method (Worksheet, Object[], int, int, bool).....	2081
Import Method (Worksheet, String[], int, int, bool).....	2083
Import Method (Worksheet, Object[,], int, int, IDataValueConverter).....	2084
Import Method (Worksheet, Int64[], int, int, bool).....	2084
Import Method (Worksheet, Decimal[], int, int, bool).....	2085
Import Method (Worksheet, Single[], int, int, bool).....	2086
Import Method (Worksheet, DateTime[], int, int, bool).....	2087
Import Method (Worksheet, String[,], int, int, DataImportOptions).....	2087
Import Method (Worksheet, Object[,], int, int, DataImportOptions).....	2088
Import Method (Worksheet, object, int, int, DataSourceImportOptions).....	2089
Import Method (Worksheet, Int16[], int, int, bool).....	2091
Import Method (Worksheet, Double[], int, int, bool).....	2092
Import Method (Worksheet, Int32[], int, int, bool).....	2093
Import Method (Worksheet, IDataReader, bool, int, int, IDataValueConverter).....	2093
Import Method (Worksheet, Object[], int, int, bool, IDataValueConverter).....	2094
Import Method (Worksheet, Object[], int, int, bool, DataImportOptions).....	2095
Import Method (Worksheet, DataTable, bool, int, int, IDataValueConverter).....	2096
Import Method (Worksheet, DataTable, bool, int, int, DataImportOptions).....	2097
Import Method (Worksheet, IEnumerable, int, int, bool, DataImportOptions).....	2098
Import Method (Worksheet, IDataReader, bool, int, int, DataImportOptions).....	2101
Import Method (Worksheet, IEnumerable, int, int, bool, IDataValueConverter).....	2101

DevExpress.Spreadsheet.Export	2103
DataTableExporterExtensions Class	2104
DataTableExporterExtensions Members	2105
DataTableExporterExtensions Methods	2106
Export Method	2107
DevExpress.UnitConversion	2110
Area Enumeration	2112
AreaUnitsConverter Class	2113
AreaUnitsConverter Members	2114
AreaUnitsConverter Constructor	2115
BaseUnitsConverter<T> Class	2116
BaseUnitsConverter<T> Members	2117
BaseUnitsConverter<T> Methods	2118
Convert Method	2119
Convert Method (Double, T, T)	2119
Convert Method (Double[], T, T)	2119
Convert Method (IList<T>, T, T)	2120
GetTransform Method	2121
BinaryPrefix Enumeration	2122
BinaryUnitsConverter Class	2123
BinaryUnitsConverter Members	2124
BinaryUnitsConverter Constructor	2125
Distance Enumeration	2126
DistanceUnitsConverter Class	2127
DistanceUnitsConverter Members	2128
DistanceUnitsConverter Constructor	2129
Energy Enumeration	2130
EnergyUnitsConverter Class	2131
EnergyUnitsConverter Members	2132
EnergyUnitsConverter Constructor	2133
Force Enumeration	2134
ForceUnitsConverter Class	2135
ForceUnitsConverter Members	2136
ForceUnitsConverter Constructor	2137
Information Enumeration	2138
InformationUnitsConverter Class	2139
InformationUnitsConverter Members	2140
InformationUnitsConverter Constructor	2141
Magnetism Enumeration	2142
MagnetismUnitsConverter Class	2143
MagnetismUnitsConverter Members	2144
MagnetismUnitsConverter Constructor	2145
Mass Enumeration	2146
MassUnitsConverter Class	2147
MassUnitsConverter Members	2148
MassUnitsConverter Constructor	2149
MetricPrefix Enumeration	2150
MetricUnitsConverter Class	2151
MetricUnitsConverter Members	2152
MetricUnitsConverter Constructor	2153
Power Enumeration	2154
PowerUnitsConverter Class	2155
PowerUnitsConverter Members	2156
PowerUnitsConverter Constructor	2157
PrefixUnitsConverter<T> Class	2158
PrefixUnitsConverter<T> Members	2159
PrefixUnitsConverter<T> Methods	2160
Convert Method	2161

Convert Method (Double, T).....	2161
Convert Method (Double[], T).....	2162
Convert Method (IList`1, T).....	2162
Convert Method (IList`1, T, T).....	2163
Convert Method (Double[], T, T).....	2163
Convert Method (Double, T, T).....	2164
GetTransform Method.....	2165
Pressure Enumeration.....	2166
PressureUnitsConverter Class.....	2167
PressureUnitsConverter Members.....	2168
PressureUnitsConverter Constructor.....	2169
QuantityValue<T> Structure.....	2170
QuantityValue<T> Members.....	2171
QuantityValue<T> Properties.....	2172
Value Property.....	2173
Speed Enumeration.....	2174
SpeedUnitsConverter Class.....	2175
SpeedUnitsConverter Members.....	2176
SpeedUnitsConverter Constructor.....	2177
Temperature Enumeration.....	2178
TemperatureUnitsConverter Class.....	2179
TemperatureUnitsConverter Members.....	2180
TemperatureUnitsConverter Constructor.....	2181
Time Enumeration.....	2182
TimeUnitsConverter Class.....	2183
TimeUnitsConverter Members.....	2184
TimeUnitsConverter Constructor.....	2185
Units Class.....	2186
Units Members.....	2187
Units Properties.....	2189
Area Property.....	2191
Binary Property.....	2192
Distance Property.....	2193
Energy Property.....	2194
Force Property.....	2195
Information Property.....	2196
Magnetism Property.....	2197
Mass Property.....	2198
Metric Property.....	2199
Power Property.....	2200
Pressure Property.....	2201
Speed Property.....	2202
Temperature Property.....	2203
Time Property.....	2204
Volume Property.....	2205
Volume Enumeration.....	2206
VolumeUnitsConverter Class.....	2208
VolumeUnitsConverter Members.....	2209
VolumeUnitsConverter Constructor.....	2210

Office File API

Office File API

The **Office File API** is a non-visual cross-platform .NET library for document processing. You can work with rich-text documents, spreadsheets or reports, draw barcodes, create new or edit existing PDF documents, convert different units and compress data directly from code, without any UI components, such as the Rich Edit Control or Spreadsheet, involved. The assembly can be used in applications targeting a variety of platforms (ASP.NET, ASP.NET MVC, WinForms and WPF).

You can explore the **Office File API** capabilities using the demo application. Refer to the [Demo Application](#) topic to learn how to find the Office File API demos and source code.



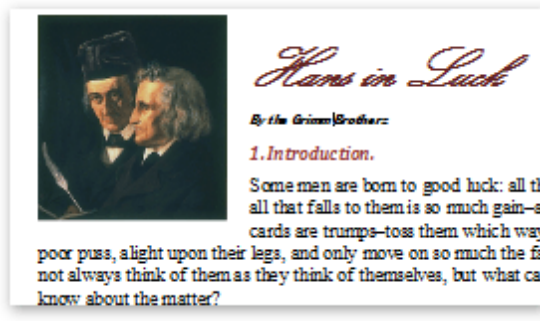
This report shows trade data on the United States on a country-by-country basis. All values are in million US\$.

Country	Exports	Imports	Balance	Exports 1Y Chg	Imports 1Y
Australia	2,277.90	815.40	1,662.50	-59.0%	
Belgium	2,808.90	1,826.20	982.70	19.2%	
Brazil	2,622.20	2,571.40	51.80	-16.0%	
Canada	26,652.10	28,262.10	-1,609.00	18.7%	
China	8,786.50	28,646.20	-27,859.70	-14.7%	
France	2,642.20	2,562.70	79.50	8.2%	
Germany	4,052.50	9,888.20	-5,835.70	8.2%	

Spreadsheet Document API

The Spreadsheet Document API is a library that allows you to create, load and modify a spreadsheet document without the use of any visual interface. The server shares all the DevExpress Spreadsheet features - editing and analyzing data, using [formulas](#) with varying difficulty levels, specifying [print settings](#) and [protecting](#) the document from editing. This component supports different [file formats](#) for both import and export.

To begin using this component, check the [Getting Started](#) topic. The complete list of examples describing how to use different Spreadsheet Document API features is available [here](#).



Word Processing Document API

As a non-visual counterpart of the DevExpress Rich Text Editor, Word Processing File API allows you to use all the word processing capabilities at runtime. The powerful API can automate such common tasks as format conversion, character and paragraph formatting, table operations, adjusting page layout options and mail merging.

Refer to the [Getting Started](#) topic to get started with this component.



PDF Document API

The PDF Document API allows you to perform various scenarios with PDF documents in code, such as merging, splitting, editing, creating, password protecting, digitally signing and much more using the straightforward API.

To get acquainted with the PDF Document API, look through the [Getting Started](#) topic.

State	Sales	Sales vs Targ Profit	Market
Alabama	449M	5.89%	1M 29%
Arizona	146M	3.68%	-1M 29%
California	173M	3.40%	10M 28%
Colorado	973M	-5.16%	-9M 20%
Connecticut	226M	0.34%	-5M 26%
Florida	188M	1.50%	9M 18%
Georgia	315M	2.09%	1M 18%

Excel Export Library

The Excel Export API provides full spreadsheet functionality, so you can make a comprehensive spreadsheet document from scratch and export it to one of the most popular Excel formats (XLSX, XLS and CSV). Unlike Spreadsheet Document Server, it writes data directly to the stream, so no in-memory model is maintained. This kind of work provides minimal memory consumption while creating a document in code.

To start with the Excel Export Library, refer to the [Getting Started](#) article. Check the [Examples](#) topic to find a way to accomplish any required task.



Snap Report API

This library provides the complete reporting engine functionality combined with the basic [Word Processing Document API](#) features. You can generate a tabular or [mail merge report](#), edit it, add visual data (sparklines, bar codes or charts) and export the resulting document to any popular format (DOC, HTML, RTF, etc.).



Zip Compression and Archive API

The Zip Compression and Archive API is a component with a zip archiver functionality. With the help of this library such tasks as generating new archives, editing existing zip files, filtering files before compressing, setting a password for the whole archive or for each file individually can be easily performed in code. To learn what steps are needed so that you can begin with the Zip Compression and Archive API, refer to the [Getting Started](#) topic. The list of examples is available [here](#).



Barcode Generation API

The Barcode Generation API allows you to generate [different types](#) of barcode images to use in the application or a document. Each barcode type provides its own [options](#) (orientation, color, quantity, etc.) that can also be specified. To get started with this library, refer to the [Getting Started](#) article.



[Unit Conversion API](#)

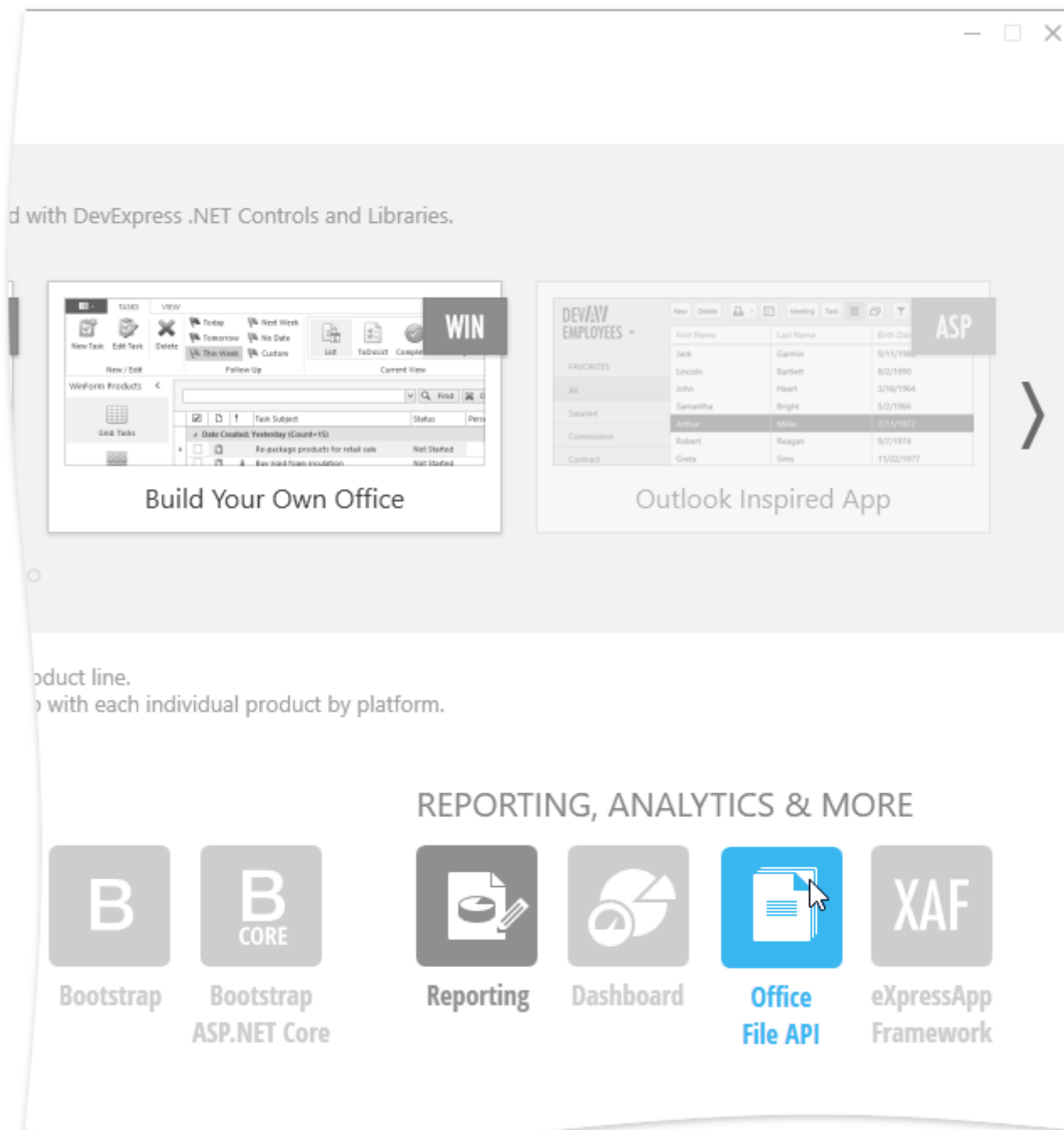
The Unit Conversion API can be helpful if you need to provide your application with an automatic conversion of different measurement units (weight, distance, power, etc.), including [values of different measurement systems](#). Learn where to start with this non-visual component [here](#).

Demo Application

[Office File API](#) > [Demo Application](#)

The DevExpress .NET Products Installer installs demos for each product. Demos are divided into two types: **Sample Applications** (real-world applications built using DevExpress products) and **Technical Demos** (which highlight a product's primary features). These demos provide an overview of each product's features, as well as implementation examples.

DevExpress Office File API demos are in the **Reporting, Analytics & More** section.



The **Office File API** demo is divided into **Winforms**, **ASP.NET** and **ASP.NET MVC**.



Launch the Demo Center

There are three ways to run the DevExpress Demo Center:

- Click the "Demo Center 18.1" shortcut in Windows Start Menu.
- In Visual Studio, select "DEVEXPRESS | Run Demo Center 18.1.3" in the Visual Studio DevExpress menu.
- Open the DevExpress product installation folder and run the Demo Center manually. The default path is C:\Users\Public\Documents\DevExpress Demos 18.1\Components\Components\Bin\DevExpress.DemoCenter.v18.1.exe".

Find the Source Code

Each demo ships with C# and VB.NET source code (ASP MVC demo ships only in C#), which describes how to solve real-world tasks using DevExpress products. Use one of the following approaches to review the required demo code:

- In the Demo Center, right-click a demo and select the "Open CS Solution"/"Open VB Solution" menu item.
- Open the demo solution manually. The default demo source directory is C:\Users\Public\Documents\DevExpress Demos 18.1\Components\Document Server\.... You can also access this folder by invoking the Windows Start Menu and clicking "All Programs" | "DevExpress 18.1" | "Demo Source Code".

Redistribution and Deployment

[Office File API](#) > [Redistribution and Deployment](#)

Certain DevExpress libraries are considered redistributable under the End User License Agreement (EULA) and can be distributed to end-users of your applications. You must have a [valid license](#) to legally distribute applications that use DevExpress components.

This topic covers the DevExpress redistribution policy and contains a list of redistributable assemblies.

Redistributable Assemblies

When you deploy a project that uses DevExpress Office File API libraries, you should copy the corresponding assembly files onto an end-user machine. The document provides the complete list of DevExpress Office File API assemblies that can be redistributed according to the terms of DevExpress EULA.

Note

In most cases, you may only use some assemblies in your project. Use the Assembly Deployment Tool to analyze your project and obtain the list of assemblies that should be deployed in your particular case.

DevExpress Assembly Deployment Tool

Deploy to:
c:\ProjectsToDeploy

Select projects to deploy:
☒ DXApplication1
☐ DXWebApplication1
☒ DXApplication2

Required redistributables:

AssembliesBy Products

☒ DevExpress.Xpf.Themes.TouchlineDark.v15.2
☒ DevExpress.Xpf.Themes.VS2010.v15.2
☒ DevExpress.XtraBars.v15.2
☒ DevExpress.XtraCharts.v15.2
☒ DevExpress.XtraCharts.v15.2.Extensions
☒ DevExpress.XtraCharts.v15.2.UI
☒ DevExpress.XtraCharts.v15.2.Web
☐ DevExpress.XtraCharts.v15.2.Wizard
☒ DevExpress.XtraEditors.v15.2
☒ DevExpress.XtraGrid.v15.2
☒ DevExpress.XtraLayout.v15.2
☒ DevExpress.XtraNavBar.v15.2
☒ DevExpress.XtraPivotGrid.v15.2
☒ DevExpress.XtraPrinting.v15.2
☒ DevExpress.XtraTreeList.v15.2
☒ DevExpress.XtraVerticalGrid.v15.2
☐ Add a custom assembly...

Reset redistributablesDeploy

The following table lists the assemblies required by applications that use the Office File API library functionality. By default, these assemblies are located in the following folder of your development machine after installation.

"C:\Program Files (x86)\DevExpress 18.1\Components\Bin\Framework\"

Assembly	Description
----------	-------------

DevExpress.Charts.v18.1.Core.dll DevExpress.XtraCharts.v18.1.dll	Contains base classes required for the charting engine.
DevExpress.Data.v18.1.dll	Implements the most basic functionality common to all DevExpress controls. This includes classes for data binding, skinning, printing, exporting, as well as many other auxiliary types and resources.
DevExpress.DataAccess.v18.1.dll	Contains classes that utilize the most popular approaches to access different data providers.
DevExpress.Docs.v18.1.dll	Contains non-visual components and object libraries for document processing.
DevExpress.Office.v18.1.Core.dll	Contains base classes common for Office File API and Office controls.
DevExpress.Pdf.v18.1.Core.dll	Contains classes that provide the basic functionality to parse and render PDF files.
DevExpress.Pdf.v18.1.Drawing.dll	Contains classes required for drawing in the PdfViewer control.
DevExpress.Printing.v18.1.Core.dll	Contains classes that implement the basic functionality for DevExpress printing libraries.
DevExpress.RichEdit.v18.1.Core.dll	Contains classes that implement the logic for formatting rich text, as well as basic types that provide a public API common to such DevExpress RTF controls as ASP.NET RichEdit, WinForms RichEdit, and WPF RichEdit. This assembly is also required when using a Rich Edit in-place editor (RepositoryItemRichTextEdit), Filter Editor Control (FilterEditorControl) and/or exporting to DOCX.
DevExpress.Snap.v18.1.Core.dll	Provides the basic functionality for Snap.
DevExpress.Spreadsheet.v18.1.Core.dll	Contains basic classes that implement the main functionality of DevExpress Spreadsheet.

See Also

The following topic describes how to use the DevExpress Assembly Deployment Tool.

- [Assembly Deployment Tool](#)

The links below cover the deployment mechanism in the .NET Framework.

- [Deploying Applications and Components](#)
- [Choosing a Deployment Strategy](#)
- [ClickOnce Deployment Overview](#)

Important

Please consult the EULA for additional up-to-the-minute information on which assemblies, tools and executables are considered redistributable.

Spreadsheet Document API

[Office File API](#) > [Spreadsheet Document API](#)

The Spreadsheet Document API is a non-visual .NET library that provides the complete spreadsheet functionality available via its API (the [Workbook](#) instance). It allows you to create, load, modify, save and print spreadsheet documents, so that you do not need Microsoft® Excel® to be installed on your computer.

The spreadsheet component also provides comprehensive data shaping and analysis tools, such as data mining, grouping, filtering, and charting. The built-in formula calculation engine allows you to create simple formulas to summarize worksheet information or construct complex formulas containing mathematical, statistical and other predefined functions for advanced data analysis.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this component or library in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

The major features of the Spreadsheet Document API are listed below.

Supported Formats

Create, load, convert and save [workbooks](#) to popular [file formats](#).

- **Excel formats:** XLS, XLSX, XLSM, XLTX, XLTM and XLT.
- **Text formats:** TXT and CSV.
- **Other formats:** PDF and HTML (export only).

Worksheet Basics

- Manage [worksheets](#): create, copy, rename, move, hide and delete worksheets.
- Customize worksheet **view options**: change the [zoom level](#) and control the visibility of [gridlines](#) and [headings](#).
- Specify [print settings](#): set paper size, margins and orientation for worksheet pages.

Cells and Cell Ranges

- Manipulate [cells and cell ranges](#): insert, copy, clear, delete, merge and unmerge cells.
- Add [formulas](#), [comments](#) and [hyperlinks](#) to cells.
- Use [defined names](#) to name individual cells and cell ranges.
- [Format cells](#): apply a cell style, or directly change cell color, alignment, borders and font settings.
- Specify cell [number format](#).
- Create [conditional formatting rules](#) to change appearance of worksheet cells based on specific conditions.

Rows and Columns

- Modify worksheet [rows and columns](#) by inserting, copying, hiding or deleting them.
- Adjust [row height and column width](#).

Data Management

- Add different [types of data](#) to worksheet cells.
- [Import data](#) to cells from different data sources (arrays, lists and data tables).

Data Binding

- Use a cell range or a [worksheet table](#) as a [data source](#) for any data-aware control.
- Establish two-way binding to various data sources so that you can further analyze or modify the data.

Formulas

- Perform calculations using [formulas](#) with different types of [cell references](#) and a comprehensive set of built-in [functions](#).
- Create [shared](#) and [array](#) formulas.
- Evaluate worksheet formulas using [Formula Engine](#), which provides the capability to parse a formula into an expression tree, modify it and rebuild the string expression.

Protection

- Protect [workbook structure](#): prevent end-users from adding, deleting or renaming worksheets.
- Protect [worksheets](#): prevent end-users from editing certain cells, applying formatting options, or making structural changes to worksheets.
- Protect [individual cell ranges](#) and unlock specific ranges in a protected worksheet for authenticated users.

Data Shaping Tools

- Organize data in [tables](#): insert, copy, modify and delete tables. Format tables using one of the built-in table styles or create your own custom styles.
- [Sort data](#) in a range in ascending or descending order.
- [Filter data](#): use the filtering functionality to arrange large amounts of data by displaying only rows that meet filtering criteria.
- [Group data](#): split data into separate groups and calculate summaries for each group.

Charts and Graphics

- Load, print and export to PDF/HTML documents containing shapes.
- Add [pictures](#) to a worksheet: insert, move, change, transform and delete pictures.
- Use the [Spreadsheet Chart API](#) to create a [basic chart](#) in code and fully [customize the appearance](#) of any chart element.
- Create [sparkline groups](#) of different types, and adjust their formatting and scaling options.

Pivot Tables

- [Create a pivot table](#) using a cell range as a data source or base your report on the data cache of the existing pivot table.
- Calculate summaries against [data fields](#) using a wide range of [aggregate functions](#) and [calculation options](#).
- Change the PivotTable [layout](#) and format a pivot table by applying a [preset](#) or [custom style](#).
- [Sort](#), [filter](#) and [group items](#) in a PivotTable report.
- Create [calculated fields](#) and [calculated items](#).

Printing and Exporting

- [Print](#) a workbook or individual worksheets to a printer or file.
- Specify various [print options](#): add headers and footers to the printout, set a print area on a worksheet, repeat specific rows and columns on every page, and define other print-related settings (such as page orientation, paper size, page margins, etc.).
- Export a workbook to a [PDF file](#), or save it as a web-page ([export to HTML](#)).

Mail Merge

- Automatically generates a set of documents based on a single [template](#) and include unique data values retrieved from a data source into each document.

See Also

[Getting Started](#)

Product Structure

[Office File API](#) > [Spreadsheet Document API](#) > [Product Structure](#)

Use the following links to access reference information about the most important classes and interfaces of the **Spreadsheet Document API**.


Class/Interface	Description
Workbook	The root object of the non-visual spreadsheet engine that allows you to create, load, edit, save and print spreadsheet documents. To use this object in production code, you would need to acquire a license for the DevExpress Office File API or DevExpress Universal Subscription.
Worksheet	An individual worksheet in a workbook.
Row	An individual row in a worksheet.
Column	An individual column in a worksheet.
Cell	A box at the intersection of a column and a row in a worksheet that can contain worksheet data, a formula and formatting.
Range	The base interface that provides the functionality required to work with a worksheet cell and a range of cells.
CellValue	A data value contained in a cell.
Formatting	An interface containing options to change cell format settings : fill, font, alignment, borders, and number format.
ConditionalFormatting	The base interface for conditional formatting rules .
DefinedName	A name that refers to a cell, cell range, formula, or constant value.
WorkbookFunctions	An interface that provides access to the built-in worksheet functions divided into categories by their functionality and allows you to override a specific function.
ICustomFunction	An interface that provides basic properties and methods required to implement a custom worksheet function .
FormulaEngine	Provides the capability to calculate and parse worksheet formulas. See Formula Engine .
Hyperlink	A hyperlink associated with a cell or cell range.
Comment	A comment attached to a worksheet cell.
Picture	An embedded image in a worksheet.
Chart	An embedded chart in a worksheet.
Table	A table in a worksheet.
PivotTable	A PivotTable report in a worksheet.
WorksheetDataBinding	An interface containing information about a cell range bound to an external data source or used to create a data source.

See Also

[Examples](#)

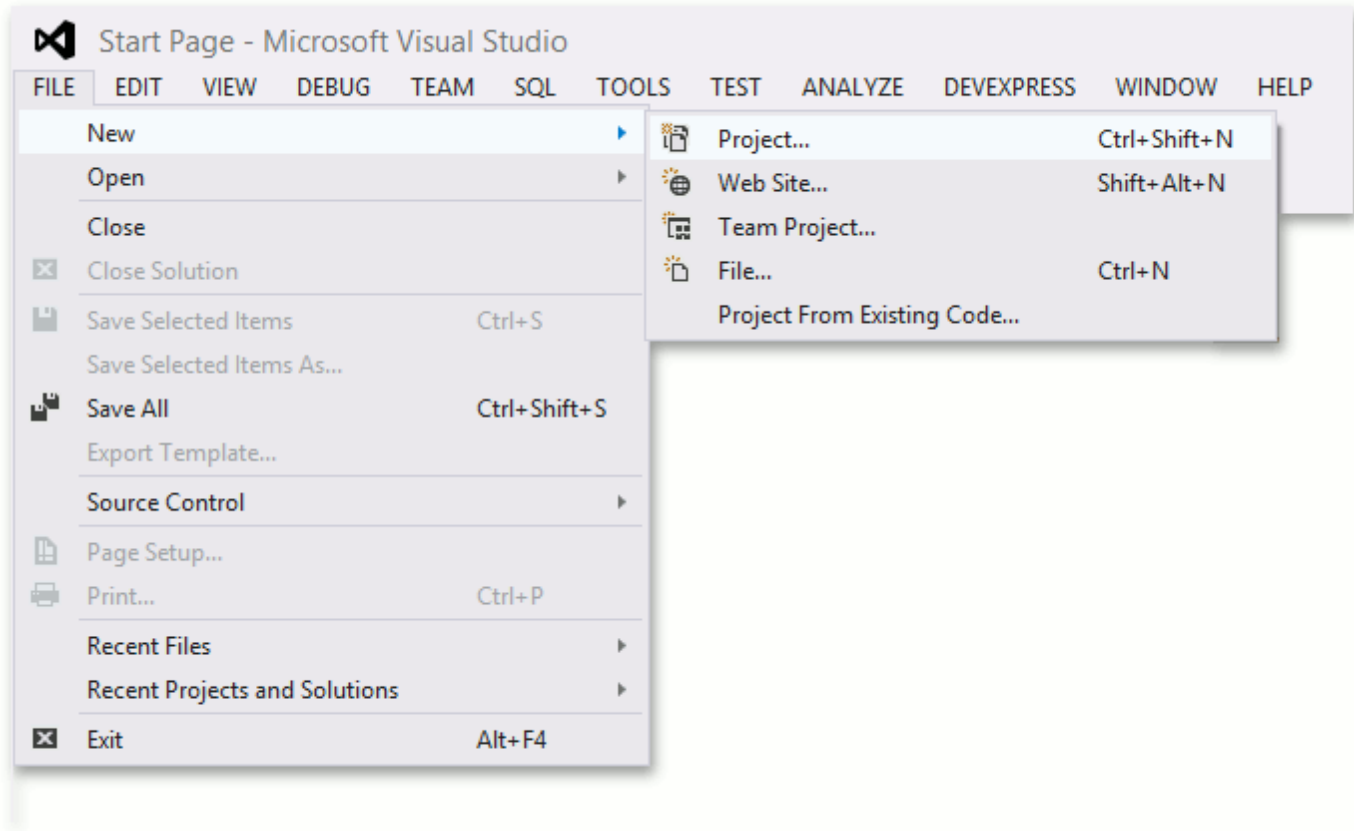
Getting Started

[Office File API](#) > [Spreadsheet Document API](#) > [Getting Started](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

- To get started with the non-visual spreadsheet component, perform the following steps.
1. Start Visual Studio and create a new Console Application project by selecting **FILE | New | Project** in the main menu. In the **New Project** window that is invoked, select **Console Application**, specify the name of the project, and click **OK**.



2. Add references to the following libraries:
 - DevExpress.Charts.v18.1.Core.dll
 - DevExpress.XtraCharts.v18.1.dll
 - DevExpress.Data.v18.1.dll
 - DevExpress.Docs.v18.1.dll
 - DevExpress.Office.v18.1.Core.dll
 - DevExpress.Spreadsheet.v18.1.Core.dll
3. Paste the code listed below in the **Main** method of the Program.cs file (**Main** procedure of the Module1.vb file for Visual Basic).

C#

```

using DevExpress.Spreadsheet;
// ...
// Create an instance of a workbook.
Workbook workbook = new DevExpress.Spreadsheet.Workbook();
// Access the first worksheet in the workbook.
Worksheet worksheet = workbook.Worksheets[0];
// Access the "A1" cell in the worksheet.
Cell cell = worksheet.Cells["A1"];
// Specify the "A1" cell value.
cell.Value = 1;
// Fill cells with sequential numbers by using shared formulas.
worksheet.Range["A2:A10"].Formula = "=SUM(A1+1)";
worksheet.Range["B1:B10"].Formula = "=A1+2";
// Multiply values contained in the cell range A1 through A10
// by the corresponding values contained in B1 through B10,
// and display the results in cells C1 through C10.
worksheet.Range["C1:C10"].ArrayFormula = "=A1:A10*B1:B10";
// Save the document file under the specified name.
workbook.SaveDocument("TestDoc.xlsx", DocumentFormat.OpenXml);
// Display gridlines in PDF.
worksheet.PrintOptions.PrintGridlines = true;
// Export the document to PDF.
workbook.ExportToPdf("TestDoc.pdf");
// Open the PDF document using the default viewer..
System.Diagnostics.Process.Start("TestDoc.pdf");
// Open the XLSX document using the default application.
System.Diagnostics.Process.Start("TestDoc.xlsx");

```

Visual Basic

```

Imports DevExpress.Spreadsheet
' ...
' Create an instance of a workbook.
Dim workbook As Workbook = New DevExpress.Spreadsheet.Workbook()
' Access the first worksheet in the workbook.
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Access the "A1" cell in the worksheet.
Dim cell As Cell = worksheet.Cells("A1")
' Specify the "A1" cell value.
cell.Value = 1
' Fill cells with sequential numbers by using shared formulas.
worksheet.Range("A2:A10").Formula = "=SUM(A1+1) "
worksheet.Range("B1:B10").Formula = "=A1+2"
' Multiply values contained in the cell range A1 through A10
' by the corresponding values contained in B1 through B10,
' and display the results in cells C1 through C10.
worksheet.Range("C1:C10").ArrayFormula = "=A1:A10*B1:B10"
' Save the document file under the specified name.
workbook.SaveDocument("TestDoc.xlsx", DocumentFormat.OpenXml)
' Display gridlines in PDF.
worksheet.PrintOptions.PrintGridlines = True
' Export the document to PDF.
workbook.ExportToPdf("TestDoc.pdf")
' Open the PDF document using the default viewer..
System.Diagnostics.Process.Start("TestDoc.pdf")
' Open the XLSX document using the default application.
System.Diagnostics.Process.Start("TestDoc.xlsx")








```



4. Run the project.

The following image shows the files generated after executing the code above.



	A	B	C	D
1	1	3	3	
2	2	4	8	
3	3	5	15	
4	4	6	24	
5	5	7		
6	6	8		
7	7	9		
8	8	10		
9	9	11		
10	10	12		
11				

File Edit View Window Help





1 / 1



I

1	3	3
2	4	8
3	5	15
4	6	24
5	7	35
6	8	48
7	9	63
8	10	80
9	11	99
10	12	120

See Also
[Training Videos](#)
[Examples](#)

Spreadsheet Document

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Document](#)

Topics in this section describe the structure of a spreadsheet document and its elements.

- [Workbook](#)
- [Worksheets](#)
- [Cells and Cell Ranges](#)
- [Rows and Columns](#)
- [Comments](#)
- [Shapes, Pictures, Charts](#)
- [Measure Units](#)

Workbook

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Document](#) > [Workbook](#)

A *workbook* is a spreadsheet document represented by the [Workbook](#) object that is a starting point for using a non-visual spreadsheet component. This class implements the IWorkbook interface, and provides a comprehensive set of properties and methods required to modify the corresponding workbook without user interaction.

Important

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

A workbook consists of one or more worksheets stored within the WorksheetCollection collection. To access this collection, use the [Workbook.Worksheets](#) property. Each worksheet has a unique [name](#) and its own [position](#) within the workbook. Thus, you can [access an individual worksheet](#) by its name or index via the WorksheetCollection.Item property.

Use other WorksheetCollection object properties to manage a workbook's set of worksheets ([add a new worksheet](#), [remove an existing worksheet](#), [assign an active worksheet](#), etc.).

A workbook may also contain specific sheets that display only a chart and do not have other data. These sheets are called **chart sheets** and are stored in the ChartSheetCollection collection returned by the [Workbook.ChartSheets](#) property. The ChartSheetCollection.Item property enables you to get an individual chart sheet by its name or index in the collection.

The [Workbook.Sheets](#) collection stores all sheets (worksheets and chart sheets) contained in a workbook. Use this collection when you need to access a sheet of any type.

By default, when you create a [Workbook](#) class instance, the workbook contains one empty worksheet ("Sheet1"). The same workbook is also created when you call the [Workbook.CreateNewDocument](#) method. To [load](#) an existing document, use the [Workbook.LoadDocument](#) method. To [save](#) a workbook after all modifications have been completed, use the [Workbook.SaveDocument](#) method. The [Supported Formats](#) topic lists the file formats supported for loading and saving workbooks.

You can also [print](#) a workbook using the [Workbook.Print](#) method overloads. The WorksheetView and Worksheet.PrintOptions properties allow you to set print options for each individual worksheet.

A workbook contains a collection of [styles](#) that can be used to format cell appearance in any worksheet. To access and modify this collection (for example, [add new styles or modify existing styles](#)), use the [Workbook.Styles](#) property.

The Worksheet.DefinedNames collection includes cell, formula and constant defined names that can be used in any worksheet within the workbook without qualification. See the [Defined Names](#) topic for more details.

The [Workbook.DocumentSettings](#) property provides access to an object whose members you can use to customize different workbook settings. For example, you can specify whether the R1C1 reference style should be used in a workbook (DocumentSettings.R1C1ReferenceStyle) or set different workbook calculation options (DocumentSettings.Calculation).

You can also specify other workbook options using the corresponding properties of the DocumentOptions object returned by [Workbook.Options](#).

- DocumentOptions.Export, DocumentOptions.Import - Specify required parameters to import or export workbooks to different file formats.
- DocumentOptions.DocumentCapabilities - Sets document restrictions.
- DocumentOptions.Save - Defines a file name and file format to be used when saving a workbook.

Resolving Performance Issues

To improve performance while applying multiple modifications to a document, wrap your code in the [Workbook.BeginUpdate-Workbook.EndUpdate](#) method calls.

When you finish working with the [Workbook](#), you are advised to call the [Workbook.Dispose](#) method to release all the resources used by the object. This will allow you to avoid memory leaks and speed up system performance. You can also operate with the [Workbook](#) instance within the **using** statement (**Using** block in Visual Basic).

See Also

[How to: Create a New Workbook](#)
[How to: Load a Document to a Workbook](#)
[How to: Save a Document to a File](#)
[How to: Export a Workbook to PDF](#)
[How to: Specify Print Settings](#)

[How to: Show a Print Preview Form for a Workbook](#)

Worksheets

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Document](#) > [Worksheets](#)

A [workbook](#) consists of *worksheets*. A worksheet is a grid of cells organized into 1,048,576 rows and 16,384 columns. Each row is numbered ("1", "2", "3", ..., "1048576"), and each column is lettered ("A", "B", "C", ..., "XFD"). Row and column headings are displayed at the left and at the top of a worksheet, respectively.

Worksheet cells can contain data of [different types](#), and [formulas](#). These cells can be formatted. Proper cell formatting improves worksheet appearance, and allows end-users to read, find and understand worksheet data more easily. When adjusting cell appearance, you can specify font, font size, character style (bold, italic or underline), text alignment, and different colors for the background and foreground. For details on formatting cells, see the [Formatting Cells](#) document as well as the examples in the [Formatting Cells](#) section.

All worksheets in a workbook are stored within the WorksheetCollection, which you can access via the [Workbook.Worksheets](#) property. To get an individual worksheet, use WorksheetCollection.Item. Each worksheet is accessible by its unique [name](#) or [position](#) within the workbook (see the [How to: Access a Worksheet](#) example).

To set the active worksheet within a workbook, use the WorksheetCollection.ActiveWorksheet property (see the [How to: Set an Active Worksheet](#) example).

The Worksheet interface represents a worksheet. It provides a wide range of members to access and manage different worksheet elements. The following table lists the main properties and methods.

Property / Method	Description	Example
Worksheet.Rows	Returns the collection of all worksheet rows .	How to: Access a Row or Column
Worksheet.Columns	Returns the collection of all worksheet columns .	How to: Access a Row or Column
Worksheet.Cells	Returns the collection of all worksheet cells . You can use this property to format the entire worksheet at once, or access individual cells.	How to: Access a Cell in a Worksheet
Worksheet.Range	Provides access to a range of cells .	How to: Access a Range of Cells
Worksheet.DefinedNames	Returns the collection of worksheet defined names associated with cells, cell ranges, formulas and constants.	How to: Create a Named Range of Cells How to: Create Named Formulas
Worksheet.Hyperlinks	Returns the collection of hyperlinks contained in worksheet cells.	How to: Add a Hyperlink to a Cell
Worksheet.Comments	Returns the collection of comments contained in worksheet cells.	How To: Add a Comment To a Cell
Worksheet.Shapes	Returns the collection of all graphics contained in the worksheet.	
Worksheet.GetUsedRange	Returns a worksheet cell range that contains data.	
Worksheet.Name	Specifies the worksheet name.	How to: Rename a Worksheet
Worksheet.Index, Worksheet.Move	Control worksheet position within a workbook.	How to: Move a Worksheet to another Location
Worksheet.Visible, Worksheet.VisibilityType	Control worksheet visibility.	How to: Show and Hide a Worksheet

Worksheet.PrintOptions	Specifies options that affect how a worksheet is printed.	How to: Specify Print Settings
------------------------	---	--

A worksheet view holds a set of display settings applied to a worksheet. To change these settings, modify the properties of the `WorksheetView` object that is accessed via the `Worksheet.ActiveView` property.

Property	Description	Example
<code>WorksheetView.ViewType</code>	Specifies the worksheet view.	
<code>WorksheetView.Orientation</code>	Specifies worksheet page orientation (landscape or portrait).	How to: Set Page Orientation
<code>WorksheetView.Margins</code>	Specifies the margins of worksheet pages. Use the Workbook.Unit property to select the unit of measure for page margins.	How to: Set Page Margins
<code>WorksheetView.PaperKind</code>	Specifies the paper size of worksheet pages when the worksheet is printed.	How to: Set Paper Size
<code>WorksheetView.ShowGridlines</code>	Controls the visibility of worksheet gridlines.	How to: Show and Hide Gridlines
<code>WorksheetView.ShowHeadings</code>	Specifies whether to show or hide row and column headings in a worksheet.	How to: Show and Hide Row and Column Headings
<code>WorksheetView.TabColor</code>	Specifies the worksheet tab color.	
<code>WorksheetView.Zoom</code>	Sets the zoom level for a worksheet view.	How to: Zoom In and Out of a Worksheet

See Also

[Worksheet Examples](#)

Cells and Cell Ranges

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Document](#) > [Cells and Cell Ranges](#)

All worksheet data is stored in *cells*. Each cell can only hold and display a single piece of data - the [cell value](#). A cell can also contain a [formula](#) that calculates the cell value dynamically.

An individual cell is a box at the intersection of a [column and row](#). Thus, a *cell reference* is usually a combination of a column letter and a row number. For example, **C3** refers to a cell that belongs to column **C** and row **3**. Other [cell reference styles](#) are also supported.

You can [format cells](#) to improve worksheet appearance, and make it easy for end-users to read and understand its content.

When working with worksheet data, you can access and modify individual cells (Cell) as well as cell ranges (Range). The Range interface provides the basic functionality required to work with worksheet cells.

Property / Method	Description	Example
CellCollection.Item, Worksheet.Item Row.Item, Column.Item, Range.Item	Access an individual cell from a worksheet's collection of all cells, from a particular row or column, or from any range of cells.	How to: Access a Cell in a Worksheet
Worksheet.Range Worksheet.GetUsedRange	When working with a worksheet (for example, formatting cells or processing data), you can manipulate not only individual cells but also cell ranges. You can access a range of contiguous cells (Range) or a cell range that includes the entire content of a worksheet (the worksheet's used range).	How to: Access a Range of Cells
Range.GetReferenceA1, Range.GetReferenceR1C1	Return a cell or cell range reference in the corresponding cell reference style .	How to: Use Cell and Worksheet References in Formulas
Cell.ColumnIndex, Cell.RowIndex Range.TopRowIndex, Range.BottomRowIndex, Range.LeftColumnIndex, Range.RightColumnIndex, Range.RowCount, Range.ColumnCount	For an individual cell, you can obtain indexes of the row and column to which this cell belongs. For a range of cells, you can get indexes of its bound rows and columns, and obtain the number of rows and columns in the range.	
Range.Value	Sets data of a specified type to a cell or a cell range. You can also use this property to retrieve information about the type of a cell's actual value, and get the cell value itself as an object of the corresponding type. If a cell contains a formula, this property returns a value resulting from the formula. For more information, see the Cell Data Types document.	How to: Change a Cell or Cell Range Value
Cell.DisplayText	Gets a cell value as it is displayed.	
Range.Formula Range.ArrayFormula, Range.HasArrayFormula, Cell.IsTopLeftCellInArrayFormulaRange	Use formulas in cells to dynamically perform calculations on worksheet data. Values displayed in cells with formulas are automatically recalculated and updated after processed data has been changed.	How to: Add Formulas to Cells How to: Create Shared Formulas How to: Create Array Formulas

Worksheet.InsertCells	Inserts empty cells in a worksheet, above or to the left of the specified cell or cell range, shifting other cells in the same row to the right and cells in the same column down.	How to: Insert a Cell or Cell Range
Range.CopyFrom	Copy information (all information, values only, formats only, borders only, etc.) from cells to other cells.	How to: Copy Cell Data Only, Cell Style Only, or Cell Data with Style
Worksheet.ClearContents Worksheet.ClearFormats Worksheet.ClearComments Worksheet.ClearHyperlinks Worksheet.Clear	Remove content, formats, hyperlinks, comments or all information from cells.	How to: Clear Cells of Content, Formatting, Hyperlinks and Comments
Range.BeginUpdateFormatting - Range.EndUpdateFormatting Range.Style Range.FillColor Formatting.Fill Formatting.Font Formatting.NumberFormat Formatting.Alignment Formatting.Borders, Range.SetInsideBorders	Format an individual cell or a cell range using styles, or by directly changing and setting the required formatting characteristics for a cell or a range of cells. For details, see Formatting Cells .	Formatting Cells
Range.IsIntersecting Range.Intersect	Allow you to determine whether or not cell ranges intersect, and obtain the intersection of ranges.	
Range.Name Worksheet.DefinedNames, DefinedNameCollection.Add	You can name individual cells and cell ranges (as well as formulas and constants), making it easier to understand the purpose of named cells, and as a result, making it easier to use them. For more information, refer to the Defined Names topic.	How to: Create a Named Range of Cells
Worksheet.DeleteCells	Deletes cells from a worksheet, shifting other cells in the same row to the left, and cells in the same column up.	How to: Delete a Cell or Range of Cells
Worksheet.Hyperlinks HyperlinkCollection.Add HyperlinkCollection.GetHyperlinks	Insert hyperlinks into cells.	How to: Add a Hyperlink to a Cell
Worksheet.Comments CommentCollection.Add CommentCollection.GetComments	Associate cells with comments .	How To: Add a Comment To a Cell
Worksheet.MergeCells, Worksheet.UnMergeCells	Merge several adjacent cells into one cell, and split merged cells.	How to: Merge Cells or Split Merged Cells

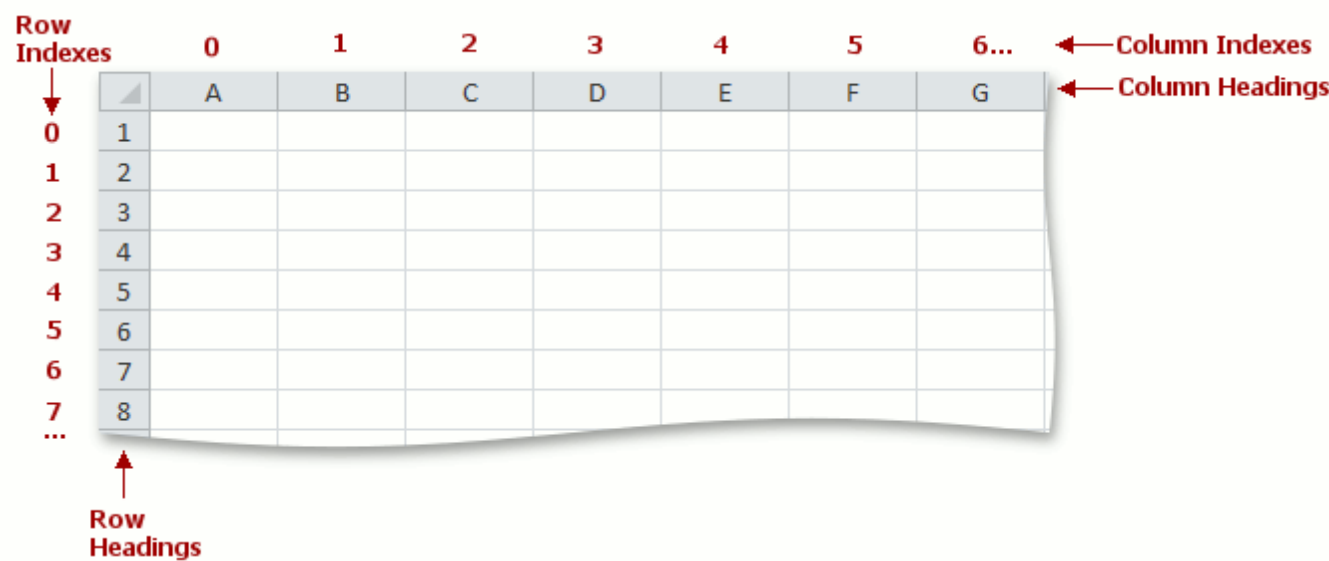
See Also
[Cell Examples](#)

Rows and Columns

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Document](#) > [Rows and Columns](#)

[Worksheet](#) cells are organized into 1,048,576 rows and 16,384 columns. The number of rows and columns is permanently fixed.

Rows and columns have headings in a worksheet (rows are numbered - "1", "2", "3",..., "1048576", and columns are lettered - "A", "B", "C",..., "XFD"). You can hide or show these row and column headings via the `WorksheetView.ShowHeadings` property.



All worksheet rows and columns are stored in the `RowCollection` and `ColumnCollection` objects that you can access via the `Worksheet.Rows` and `Worksheet.Columns` properties. The `Row` and `Column` objects specify an individual row or column, respectively. Use the properties and methods of these objects to manage worksheet rows and columns.

Property / Method	Description	Example
<code>RowCollection.Item</code> <code>ColumnCollection.Item</code>	Provide access to an individual row or column by its heading (<code>Row.Heading</code> or <code>Column.Heading</code>) or index (<code>Row.Index</code> or <code>Column.Index</code>).	How to: Access a Row or Column
<code>RowCollection.Insert</code> , <code>Row.Insert</code> <code>ColumnCollection.Insert</code> , <code>Column.Insert</code>	Insert new rows and columns into a worksheet.	How to: Add a New Row or Column to a Worksheet
<code>Range.CopyFrom</code>	Copies a row or column.	How to: Copy a Row or Column
<code>RowCollection.Remove</code> , <code>Row.Delete</code> <code>ColumnCollection.Remove</code> , <code>Column.Delete</code>	Remove specified rows and columns from a worksheet.	How to: Delete a Row or Column from a Worksheet
<code>RowCollection.Group</code> , <code>RowCollection.UnGroup</code> <code>ColumnCollection.Group</code> , <code>ColumnCollection.UnGroup</code> <code>Row.GroupLevel</code> , <code>Column.GroupLevel</code>	Group and ungroup rows and columns in a worksheet.	Grouping
<code>Row.Visible</code> <code>Column.Visible</code>	Control row or column visibility in a worksheet.	How to: Hide a Row or a Column
<code>Row.Height</code> , <code>Range.RowHeight</code> , <code>Worksheet.DefaultRowHeight</code>	Specify row height and column width in a worksheet.	How to: Specify Row Height or Column Width

Column.Width, Column.WidthInCharacters, Column.WidthInPixels Range.ColumnWidth, Range.ColumnWidthInCharacters Worksheet.DefaultColumnWidth, Worksheet.DefaultColumnWidthInCharacters, Worksheet.DefaultColumnWidthInPixels		
---	--	--

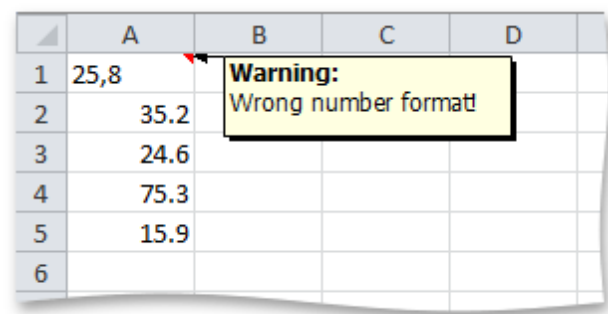
See Also
[Row and Column Examples](#)

Comments

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Document](#) > [Comments](#)

A **comment** is a note that you attach to a cell, separate from other cell content. Comments can make a [worksheet](#) easier to understand by providing additional context for the data it contains.

The image below illustrates how a comment is displayed in Microsoft® Excel®.



A [worksheet](#) stores comments in the CommentCollection object that is accessed via the Worksheet.Comments property. The CommentCollection interface provides the basic methods to work with comments. For example, to add a comment to a cell, use the CommentCollection.Add method (see [How To: Add a Comment To a Cell](#) example). To delete an individual comment from the collection, use the CommentCollection.Remove or CommentCollection.RemoveAt method. To delete all comments from the worksheet, use CommentCollection.Clear method.

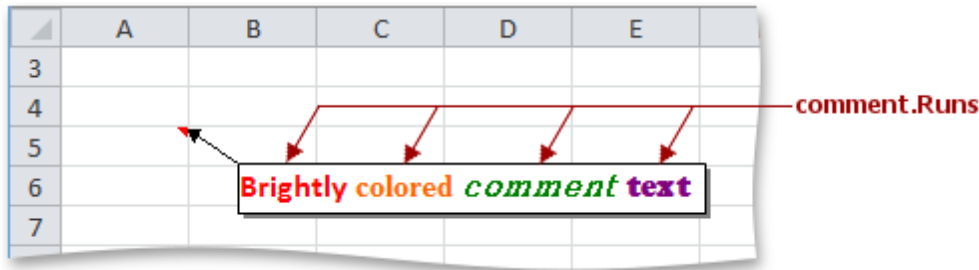
Each comment is represented by the Comment object, which provides the following properties to specify a comment.

Property	Description
Comment.Author	Gets or sets the author of the comment.
Comment.Text	Gets or sets the comment text.
Comment.Runs	Provides access to the comment text regions, each of which has specific formatting.
Comment.Reference	Returns an absolute reference to the commented cell.
Comment.Visible	Specifies whether the comment should be displayed in a cell.

Comment Runs

Comment text is grouped into one or more **runs**. Each run is specified by the CommentRun object and defines a region of the comment text with its own set of font characteristics. Use the CommentRun.Text property to define the text region within a comment, to which you want to apply specific formatting (CommentRun.Font).

The comment runs are stored in the CommentRunCollection collection that is accessed via the Comment.Runs property.



When you create a new comment via the CommentCollection.Add method or set the Comment.Text of an existing comment, the CommentRunCollection collection includes a single run that holds the full text of the comment formatted with the default font. You can access this run object and change the font to be applied to the comment text.

The `CommentRunCollection` interface provides methods to operate with comment runs. To add a new comment run, use the `CommentRunCollection.Add` method (see [How To: Add a Comment To a Cell](#) example). To delete the specified comment run from the collection, use the `CommentRunCollection.Remove` or `CommentRunCollection.RemoveAt` method. To delete all text from the comment, use the `CommentRunCollection.Clear` method.

See Also

[How To: Add a Comment To a Cell](#)

[How to: Clear Cells of Content, Formatting, Hyperlinks and Comments](#)

Shapes, Pictures, Charts

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Document](#) > [Shapes, Pictures, Charts](#)

A worksheet may contain embedded drawing objects - shapes, pictures and charts. An image in a worksheet is a Picture object, and an embedded chart is a Chart object. Both these interfaces are inherited from the base Shape interface. Other shape types (ShapeType.Shape, ShapeType.Connector and ShapeType.Group) cannot be currently accessed in code.

Accessing Drawing Objects

Drawing objects are contained in the ShapeCollection collection. The collection is accessible using the Worksheet.Shapes property and contains all pictures and charts in a worksheet. The Worksheet.Pictures property gets a collection of all pictures, and the Worksheet.Charts property gets a collection of all charts in a worksheet.

Drawing Object Type

Picture or chart - the object's type can be determined by its Shape.ShapeType property. If the type is ShapeType.Picture, you can cast the object to the Picture type and use the Picture.Image property to obtain the object's image. If the type is ShapeType.Chart, you can cast the object to the Chart type.

Location

A drawing object position in a worksheet is determined by the following properties.

Property	Description
FloatingObject.Left	Gets or sets the distance from the left edge of the worksheet to the top left corner of the floating object.
FloatingObject.OffsetX	Gets a distance from the top left corner of the floating to the left edge of the cell where the top left corner of the floating object is located.
FloatingObject.Top	Gets or sets the distance from the top edge of the worksheet to the top left corner of the floating object.
FloatingObject.OffsetY	Gets a distance from the top left corner of the floating object to the top edge of the cell where the top left corner of the floating object is located.
FloatingObject.TopLeftCell	Gets or sets a cell where the top left corner of a floating object is located.
FloatingObject.BottomRightCell	Gets or sets a cell where the bottom right corner of a floating object is located.
Shape.ZOrderPosition	Gets the position of the current drawing object in the z-order.

Use the FloatingObject.Move method to change the location.

Size and Rotation

Utilize the FloatingObject.Height and FloatingObject.Width to specify the drawing object's size. If the Shape.LockAspectRatio option is set to **true**, the modification of one property results in another property being changed to retain the aspect ratio.

Use the Shape.Rotation property to set the angle to which a drawing object is rotated, and use the Shape.IncrementRotation method to rotate the drawing object by a specified number of degrees.

Placement

A drawing object may respond when cells underneath it are moved or resized. This behavior is determined by the FloatingObject.Placement property. A drawing object can be moved and resized with cells, or it can float freely.

When a picture is added to a worksheet, its initial placement depends on the specific overload of the `PictureCollection.AddPicture` method. If a `Cell` is passed to the method, the picture will move with cells. If a `Range` is passed to the method, the picture will move and size with cells. If coordinates are specified when an image is inserted in a worksheet, it will float freely.

To move a drawing object in front of or behind other objects in a worksheet, specify its `Shape.ZOrderPosition` property relative to other drawing objects. When drawing objects are added to the `Worksheet.Shapes` collection, the `ZOrderPosition` gets its value incremented by one starting from 1. You can increase the **`ZOrderPosition`** property value to move the drawing object in front of all objects with lower `ZOrderPosition`. The `ShapeCollection.NormalizeZOrder` method modifies `ZOrderPosition` values for all drawing objects in a collection, so that they become a series of consecutive integers starting from 1.

Hyperlink

Use the `Shape.InsertHyperlink` method to create a hyperlink for a drawing object. Use the `Shape.RemoveHyperlink` method to delete a hyperlink.

See Also

[Charts](#)

[Pictures](#)

Measure Units

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Document](#) > [Measure Units](#)

You can select a single unit of measurement to specify various distance values of a workbook (such as column width, row height and page margins). To do this, set the [Workbook.Unit](#) property.

Unit of Measurement	Description	How to Use
Document	Document (1 document unit is equal to 1/300 of an inch). This is a default measure unit of a workbook.	Set the Unit property to DocumentUnit.Document.
Centimeter	Centimeter (1 inch is equal to 2.54 centimeters)	Set the Unit property to DocumentUnit.Centimeter.
Millimeter	Millimeter (10 millimeters are equal to 1 centimeter)	Set the Unit property to DocumentUnit.Millimeter.
Inch	Inch	Set the Unit property to DocumentUnit.Inch.
Point	Point (1 inch is equal to 72 points)	Set the Unit property to DocumentUnit.Point.
Pixel	Pixel (1 inch is equal to 96 pixels). You can use this unit to set column width.	Use the following properties to set column width in pixels. Worksheet.DefaultColumnWidthInPixels Column.WidthInPixels
Character	Character (the width of the zero character in the font specified by the built-in Normal style). You can use this unit to set column width.	Use the following properties to set column width in characters. Worksheet.DefaultColumnWidthInCharacters Range.ColumnWidthInCharacters Column.WidthInCharacters

A workbook's units of measurement specified by the **Unit** property are used in the following cases.

- When you specify a column width via the Worksheet.DefaultColumnWidth, Range.ColumnWidth or Column.Width property.
- When you specify a row height via the Worksheet.DefaultRowHeight, Range.RowHeight or Row.Height property.
- When you specify page margins via the properties of the Margins object.

See Also

[How to: Specify Row Height or Column Width](#)

[How to: Set Page Margins](#)

Supported Formats

[Office File API](#) > [Spreadsheet Document API](#) > [Supported Formats](#)

The non-visual spreadsheet component supports the following file formats to import and export data.

- **XLSX**

Microsoft Office Open XML format - the default file format of Microsoft Excel, starting with Microsoft Excel 2007.

- **XLSM**

Microsoft Office Open XML format with macro support (limited support - macros cannot be executed or modified)

- **XLS**

Microsoft Excel 97-2003 binary file format.

- **XLTX**

Microsoft Office Open XML template file.

- **XLTM**

Microsoft Office Open XML macro-enabled template file (limited support - macros cannot be executed or modified).

- **XLT**

Microsoft Excel 97-2003 template file.

- **CSV**

Comma Separated Values - the plain text format using comma characters as separators between cells.

- **TXT**

Tab Delimited Text - the plain text format using tab characters as separators between cells.

- **PDF (export only)**

Portable Document Format.

- **HTML (export only)**

Web Page.

Cell Basics

[Office File API](#) > [Spreadsheet Document API](#) > [Cell Basics](#)

This section consists of the following topics.

- [Cell Data Types](#)
- [Dates and Times in Cells](#)
- [Error Types](#)
- [Cell Referencing](#)
- [Formatting Cells](#)

Cell Data Types

[Office File API](#) > [Spreadsheet Document API](#) > [Cell Basics](#) > [Cell Data Types](#)

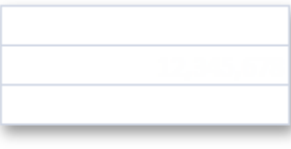

Each [cell](#) in a worksheet has a value that is specified by the `CellValue` object. To access this object, use the `Range.Value` property. A cell value is determined by the data contained within the cell:

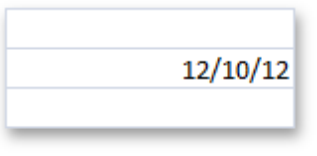
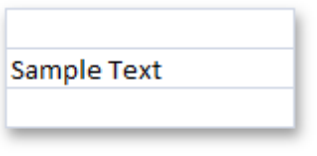
- If the cell does not contain any data, the cell value is empty (`CellValue.IsEmpty` is set to **true**). Refer to the [How to: Clear Cells of Content, Formatting, Hyperlinks and Comments](#) example to learn how to remove cell contents.
- The cell value is determined by a constant assigned to the cell via the `Range.Value` property. In this case, the cell value is neither calculated nor changed.
- The cell value is determined by a value resulting from a [formula](#) assigned to a cell via the `Range.Formula` property. In this case, the cell value is calculated dynamically.

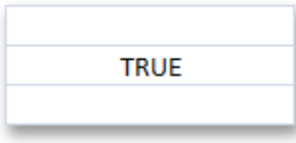

A cell value can be of one of the following types: **empty**, **numeric**, **text**, **Boolean** or **error**. Cell values may have various display formats. For example, a numeric value can be displayed as a decimal number, a percentage or currency value, a date or time value, etc.

Use the properties of the `CellValue` object returned by the `Range.Value` property to retrieve information about the cell value type, and get the cell value itself as an object of the corresponding type. To get the string specifying the formatted value as it is displayed in a cell, use the `Cell.DisplayText` property.

The table below lists the available cell value types and provides examples on how a value of each type can be input, formatted, displayed and obtained.

Cell Value Type	Cell Content	Sample Input	Sample Display Format	Displayed String (Cell.DisplayText)	Identify the Type	Obtain the Cell Value Object
Empty	The default cell value type. If a cell contains any data, you can assign an empty value to it by setting the <code>Range.Value</code> property to null or <code>CellValue.Empty</code> . For details, refer to the How to: Clear Cells of Content, Formatting, Hyperlinks and Comments document.	<code>cell.Value = null</code> - or - <code>cell.Value = CellValue.Empty</code>			<code>CellValue.IsEmpty = true</code> <code>CellValue.Type = CellValueType.None</code>	<code>CellValue.Empty</code>
Numeric	The cell's <code>Range.Value</code> property is assigned to a value of any numeric type (e.g., Int32 ,	<code>cell.Value = 12345678</code> - or - <code>cell.Formula = "=SUM(12000, 000,345678)"</code>	<code>cell.NumberFormat = "#,#"</code> More examples: How to: Specify Number or Date		<code>CellValue.IsNumeric = true</code> <code>CellValue.Type = CellValueType.Numeric</code>	<code>CellValue.NumericValue</code>

	<p>Double, etc.).</p> <p>- or -</p> <p>The cell's Range.For mula property is assigned to an expression that returns a number.</p>		<p>Format for Cell Content</p>			
<p>Numeric (Date and Time)</p>	<p>The cell's Range.Valu e property is assigned to the DateTime object, an object returned by the CellValue.F romDateTi me method, or a numeric value representing a serial number of a date or time.</p> <p>- or -</p> <p>The cell's Range.For mula property is assigned to an expression that returns the serial number of a date or time.</p> <p>For details, refer to the Dates and Times in Cells document.</p>	<p>cell.Value = new DateTime(2012, 12, 10);</p> <p>- or -</p> <p>workbook. Documents ettings.Cal culation.Us e1904Date System = true; cell.Value = CellValue.F romDateTi me(new DateTime(2012, 12, 10), true);</p> <p>- or -</p> <p>cell.Value = 41253;</p> <p>- or -</p> <p>cell.Formul a = "=DATE(20 12,12,10)";</p>	<p>cell.Numbe rFormat = "m/d/yy"</p> <p>More examples:</p> <p>How to: Specify Number or Date Format for Cell Content</p>		<p>CellValue.I sNumeric = true</p> <p>CellValue.T ype = CellValueT ype.DateTi me</p> <p>CellValue.I sDateTime = true</p>	<p>CellValue.D ateTimeVal ue</p> <p>CellValue.N umericValu e</p>
<p>Text</p>	<p>The cell's Range.Valu e property is assigned to the String object.</p> <p>- or -</p>	<p>cell.Value = "Sample Text"</p> <p>- or -</p> <p>cell.Formul a = "= PROPER("s ample text")"</p>			<p>CellValue.I sText = true</p> <p>CellValue.T ype = CellValueT ype.Text</p>	<p>CellValue.T extValue</p>

	<p>The cell's Range.For mula property is assigned to an expression that returns text.</p>					
Boolean	<p>The cell's Range.Valu e property is assigned to the Boolean object.</p> <p>- or -</p> <p>The cell's Range.For mula property is assigned to an expression that returns TRUE or FALSE.</p>	<p>cell.Value = true</p> <p>- or -</p> <p>cell.Formul a = "= TRUE()"</p>			<p>CellValue.I sBoolean = true</p> <p>CellValue.T ype = CellValueT ype.Boolea n</p>	<p>CellValue.B ooleanValu e</p>
Error	<p>The cell's Range.Valu e property is assigned to the CellValue object returned by the CellValue.Error* field (for example, CellValue.E rrorDivisio nByZero, CellValue.E rrorInvalid ValueInFun ction, CellValue.E rrorName, etc.).</p> <p>- or -</p> <p>The cell's Range.For mula property is assigned to an error code (e.g., "= #DIV/0!", "= #N/A", etc.) or an expression that cannot</p>	<p>cell.Value = CellValue.E rrorDivisio nByZero</p> <p>- or -</p> <p>cell.Formul a = "= #DIV/0!"</p> <p>- or -</p> <p>cell.Formul a = "=5/0"</p>			<p>CellValue.I sError = true</p> <p>CellValue.T ype = CellValueT ype.Error</p>	<p>CellValue.E rrorValue</p>

	be calculated correctly (for example, an expression containing an invalid function name, value, division by zero, etc.) For details, refer to the Error Types document.					
--	---	--	--	--	--	--

See Also

- [How to: Change a Cell or Cell Range Value](#)
- [Dates and Times in Cells](#)
- [Error Types](#)

Dates and Times in Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Cell Basics](#) > [Dates and Times in Cells](#)

Dates and times are stored in [cells](#) as numbers. Thus, the values of cells that contain dates and times are of the [numeric type](#). A number that specifies a date and time consists of the [date](#) (integer part) and [time](#) (fractional part) components. The `CellValue.NumericValue` property returns this number.

To display a number as a date and time, apply the required [date and time format](#) to a cell via the `Formatting.NumberFormat` property. The `CellValue.DateTimeValue` property returns the [DateTime](#) object, which specifies the date and time that is represented by the number contained in a cell.

	A	B	C
1	Numeric Value	Date and Time Format	Formatted Numeric Value
2	41439.48239	m/d/yy h:mm AM/PM	6/14/13 11:34 AM
3			
4			

To obtain whether or not a cell displays its numeric value as date and time, use the `Cell.IsDisplayedAsDateTime` property.

Dates in Cells

`SpreadsheetControl` stores dates as numbers that are called *serial values*.

A serial value is an integer that is the number of elapsed days from the first day in the date system. `SpreadsheetControl` supports the following date systems for serial values:

- *The 1900 date system.* The first date is January 1, 1900, and its serial value is 1. The last date is December 31, 9999, and its serial value is 2,958,465.

This date system is used in the workbook by default.

- *The 1904 date system.* The first date is January 1, 1904, and its serial value is 0. The last date is December 31, 9999, and its serial value is 2,957,003.

To use this date system in the workbook, set the **`Workbook.DocumentSettings.Calculation.Use1904DateSystem`** property to **true**.

The serial value of the date contained in a cell is the integer part of the number that the `CellValue.NumericValue` property returns. You can input a date into a cell in one of the following ways:

- Set the `Range.Value` property to a number that specifies the date's serial value in the 1900 or 1904 date system.
- Set the `Range.Value` property to a [DateTime](#) object. The date's serial value will be calculated in the 1900 date system.
- Set the `Range.Value` property to an object that is returned by the `CellValue.FromDateTime` method. The date's serial value will be calculated based on the passed *use1904DateSystem* parameter value.
- Set the `Range.Formula` property to an expression that returns a date. The date's serial value will be calculated according to the date system used in the workbook.

To obtain the date that corresponds to the serial value contained in a cell, use the `CellValue.DateTimeValue` property. This date is defined according to the date system used in the workbook. In other words, the `CellValue.DateTimeValue` property specifies the date that will be displayed in a cell when you [apply a date format](#).

For example, consider the 12/17/2012 date. The table below demonstrates which serial values this date is represented by in each date system, how the date can be assigned to a cell in each date system, and how the date displayed depends on the date system applied in the workbook.

Date System to Input Date	Serial Value of 12/17/2012	Input Date	Use1904DateSystem = false	Use1904DateSystem = true
1900	41260	cell.Value = new DateTime(2012, 12, 17)	CellValue.NumericValue = 41260 CellValue.DateTimeValue = {12/17/2012 12:00:00 AM}	CellValue.NumericValue = 41260 CellValue.DateTimeValue = {12/18/2016 12:00:00 AM}

1904	39798	cell.Value = CellValue.FromDateTi me(new DateTime(2012, 12, 17), true);	CellValue.NumericValu e = 39798 CellValue.DateTimeVal ue = {12/16/2008 12:00:00 AM}	CellValue.NumericValu e = 39798 CellValue.DateTimeVal ue = {12/17/2012 12:00:00 AM}
------	-------	---	---	---

Times in Cells

SpreadsheetControl stores times as decimal fractions that range from 0 to 0.99999, which represent the time range from 12:00:00 AM to 11:59:59 PM. The cell's time value is the fractional part of the number that the CellValue.NumericValue property returns.

To obtain the time that corresponds to the decimal fraction of the cell's CellValue.NumericValue number, use the CellValue.DateTimeValue property. This time will be displayed in a cell when you [apply a date format](#).

Date and Time Display Formats

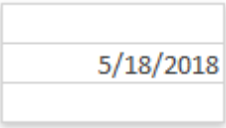
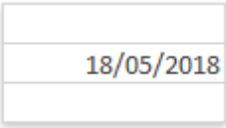
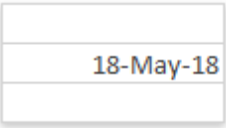
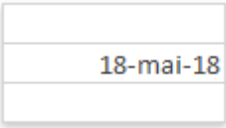
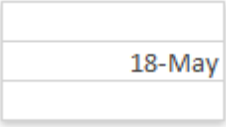

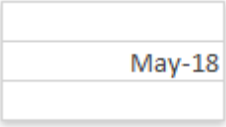
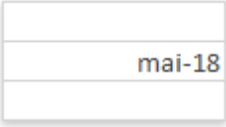
To display a number as a date and time, associate the appropriate display format with the cell via the Formatting.NumberFormat property. Use the Range.BeginUpdateFormatting-Range.EndUpdateFormatting method pair to access and modify the Formatting.NumberFormat for the cell range. To apply a number format to a single cell, use the cell's **NumberFormat** property directly. Refer to the [How to: Specify Number or Date Format for Cell Content](#) example.

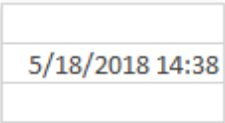
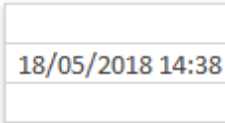
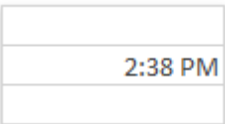
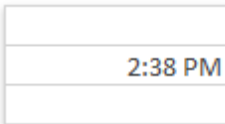
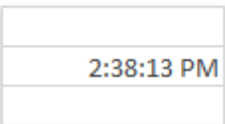
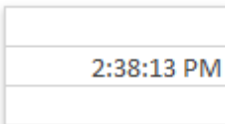

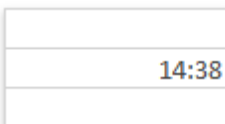
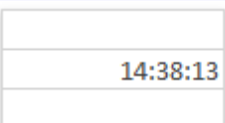
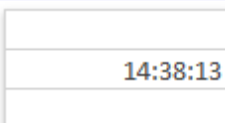
The Cell.DisplayText property returns the value as it is displayed in a cell.

Note

If a number is negative or excessively large (greater than the last date in the date system used), it will be displayed as ##### in the cell when you apply the date and time format.

When you format date and time values in a cell, you can specify display formats that rely on the operating system's regional date and time settings. The table below lists these formats in the **Invariant Culture** column. When one of the specified formats is applied to a cell, its date and time value displays differently depending on the user's current locale. The last two table columns show how the specified formats are interpreted in the English (United States) ("en-US") and French ("fr-FR") cultures.

Invariant Culture	English Culture (United States)	French Culture
"mm-dd-yy" cell.Value = DateTime.Now; cell.NumberFormat = "mm-dd-yy"	"m/d/yyyy" 	"jj/mm/aaaa" 
"d-mmm-yy" cell.Value = DateTime.Now; cell.NumberFormat = "d- mmm-yy"	"d-mmm-yy" 	"jj-mmm-aa" 
"d-mmm" cell.Value = DateTime.Now; cell.NumberFormat = "d- mmm"	"d-mmm" 	"jj-mmm" 
"mmm-yy" cell.Value = DateTime.Now; cell.NumberFormat = "mmm-yy"	"mmm-yy" 	"mmm-aa" 

"m/d/yy h:mm" cell.Value = DateTime.Now; cell.NumberFormat = "m/d/yy h:mm"	"m/d/yyyy h:mm" 	"jj/mm/aaaa hh:mm" 
"hh:mm AM/PM" cell.Value = DateTime.Now; cell.NumberFormat = "hh:mm AM/PM"	"h:mm AM/PM" 	"h:mm AM/PM" 
"hh:mm:ss AM/PM" cell.Value = DateTime.Now; cell.NumberFormat = "hh:mm:ss AM/PM"	"h:mm:ss AM/PM" 	"h:mm:ss AM/PM" 
"h:mm" cell.Value = DateTime.Now; cell.NumberFormat = "h:mm"	"h:mm" 	"hh:mm" 
"h:mm:ss" cell.Value = DateTime.Now; cell.NumberFormat = "h:mm:ss"	"h:mm:ss" 	"hh:mm:ss" 

See Also
[How to: Change a Cell or Cell Range Value](#)
[How to: Specify Number or Date Format for Cell Content](#)

Error Types

[Office File API](#) > [Spreadsheet Document API](#) > [Cell Basics](#) > [Error Types](#)

If a [cell](#) contains a formula that cannot be calculated correctly, the cell's value (Range.Value) is of the error type (the CellValue.IsError property value is **true**). To access information on an error contained in a cell, use the members of the ErrorValueInfo object returned by the CellValue.ErrorValue property. The ErrorValueInfo.Type returns the ErrorType enumeration member that specifies the error type, the ErrorValueInfo.Name property returns the error name displayed in a cell, and the ErrorValueInfo.Description property returns the cause of the error.

Error Type	Error Name	Error Description	Example
ErrorType.DivisionByZero	#DIV/0!	Division by zero!	cell.Formula = "= 10/0" cell.Formula = "= A1/B2", the B2 cell is blank
ErrorType.Name	#NAME?	Function does not exist.	cell.Formula = "= FALS" - The function name used in the formula is not spelled correctly. cell.Formula = "=SUM(A1B2)" - A colon (:) is missing in the cell range reference.
ErrorType.NotAvailable	#N/A	The value is not available to a function or formula.	cell.Formula = "= NA()" cell.ArrayFormula = "=SUM(A1:A5*A1:B3)" - An array formula 's arguments are arrays consisting of different numbers of elements.
ErrorType.Null	#NULL!	The specified intersection includes two ranges that do not intersect.	cell.Formula = "=SUM(A1:A5 E6:E8)" - The specified ranges do not intersect, so the sum cannot be calculated.
ErrorType.Number	#NUM!	Invalid numeric values in a formula or function.	cell.Formula = "=SQRT(-16)" - The square root of a negative number cannot be calculated.
ErrorType.Reference	#REF!	Cell reference is not valid.	A formula uses a reference to a cell, and then the column containing this cell is deleted: cell.Formula = "=5+D2"; worksheet.Columns["D"].Delete();
ErrorType.Value	#VALUE!	The value used in the formula is of the wrong data type.	cell.Formula = @"=SUM(""text"", 6)" - The SUM function requires numeric arguments. A formula refers to a blank cell that is not actually empty (it contains an empty string): worksheet["A1"].Value = ""; cell.Formula = "= A1 + 5";

			To fix the error in this case, specify an empty value for a cell as shown in the How to: Clear Cells of Content, Formatting, Hyperlinks and Comments example.
--	--	--	---

See Also
[How to: Change a Cell or Cell Range Value](#)

Cell Referencing

[Office File API](#) > [Spreadsheet Document API](#) > [Cell Basics](#) > [Cell Referencing](#)

A *cell reference* is a set of coordinates that specify the position of a [cell](#) or [cell range](#) on a [worksheet](#). Use cell references in [formulas](#) to obtain and process data contained in the corresponding cells. The results of formulas that use cell references are automatically updated each time the values of these cells are changed.

[Spreadsheet Document API](#) supports the following cell reference types.

- [A1 Cell References](#)

A cell is referred to by a column letter and row number.

- [R1C1 Cell References](#)

When this cell reference style is enabled, both rows and columns are numbered on a worksheet. A cell is referred to by a row number preceded by "R" and a column number preceded by "C".

- [Cross-Worksheet Cell References](#)

References to cells located in other worksheets.

- [3D Cell References](#)

References to the same cells located on multiple worksheets within a workbook.

- [External Cell References](#)

References to cells located in other workbooks.

- [Structured Cell References](#)

References to tables and table data ranges.

A1 Cell References

The A1 cell reference is a combination of column and row headings to which the cell belongs - a column letter is followed by a row number. By default, column and row headings are displayed at the top and at the left of a worksheet (see [How to: Show and Hide Row and Column Headings](#)).

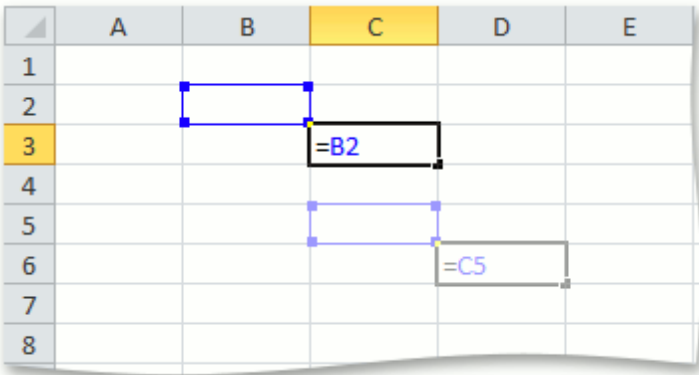
The following table provides examples of A1 references to a single cell, and various cell ranges (including references to entire rows and columns).

Cell Reference	Description
E7	Refers to a single cell located at the intersection of column E and row 7.
A2:B5	Refers to a range that includes cells from the top left cell A2 to the bottom right cell B5.
B:B	Refers to a range that includes the entire column B.
B:H	Refers to a range that includes the entire columns from B to H.
5:5	Refers to a range that includes the entire row 5.
5:10	Refers to a range that includes the entire rows from 5 to 10.

When used in a formula, a cell reference can be [relative](#), [absolute](#) or [mixed](#), depending on whether or not it should be automatically adjusted when the formula is copied.

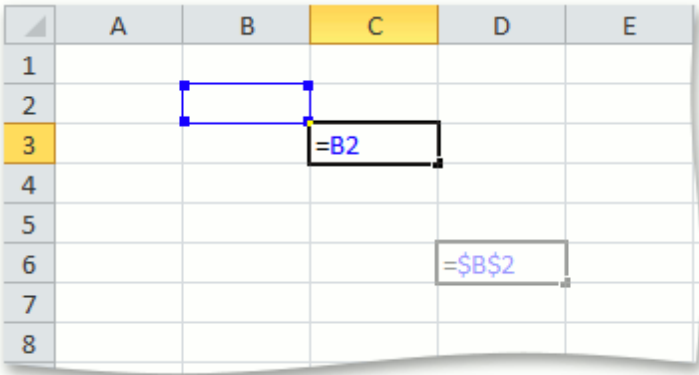
- **Relative Cell References**

A relative cell reference in a formula is based on the relative position of a referenced cell and a cell containing a formula. To keep this relative position unchanged, the cell reference is automatically changed each time you copy a formula to another cell. For example, if you copy a formula with a relative reference to cell B2 from cell C3 to D6, the reference will automatically change from B2 to C5.



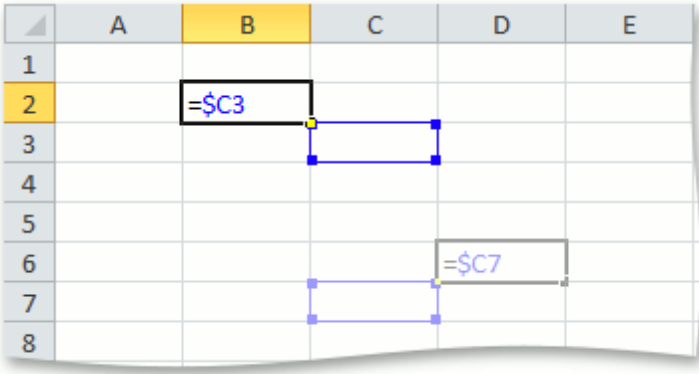
• **Absolute Cell References**

An absolute cell reference in a formula always refers to a specific cell, and it does not change if the formula is copied or moved to another cell. In absolute cell references, the column letter and row number are preceded by the '\$' sign. For example, if you copy a formula with an absolute reference to cell B2 from cell C3 to D6, the reference will remain unchanged (\$B\$2).



• **Mixed Cell References**

A mixed cell reference in a formula can either be combined from an absolute reference to a cell column and a relative reference to a cell row (for example, \$B2), or from a relative reference to a cell column and an absolute reference to a cell row (for example, A\$1). If the formula is copied, the absolute element of the mixed reference (the column letter or row number preceded by the '\$' sign) will remain unchanged, and the relative element of the reference will automatically be adjusted. For example, if you copy a formula with the \$C3 mixed reference from cell B2 to D6, the reference will change to \$C7.



R1C1 Cell References

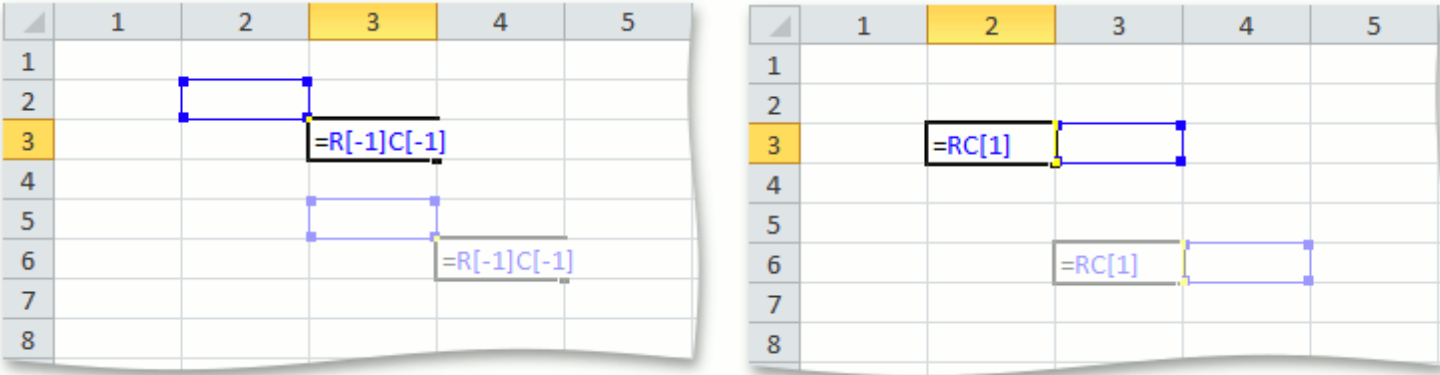
You can also use the R1C1 reference style, where both rows and columns are numbered in a worksheet. To switch on this style in a workbook, use the DocumentSettings.R1C1ReferenceStyle property. A cell reference of the R1C1 style is a combination of "R" followed by the row number and "C" followed by the column number.

Cell Reference	Description
----------------	-------------

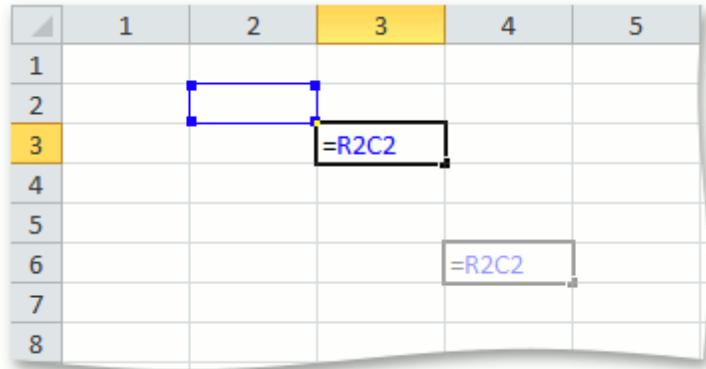
R5C7	Refers to a single cell located at the intersection of row 5 and column 7.
R3C4:R7C10	Refers to a range that includes cells from the top left cell R3C4 to the bottom right cell R7C10.
C4	Refers to a range that includes the entire column 4.
C5:C7	Refers to a range that includes the entire columns from 5 to 7.
R10	Refers to a range that includes the entire row 10.
R12:R15	Refers to a range that includes the entire rows from 12 to 15.

Like the A1 cell reference style, the R1C1 cell reference style also supports the [relative](#), [absolute](#) and [mixed](#) reference types.

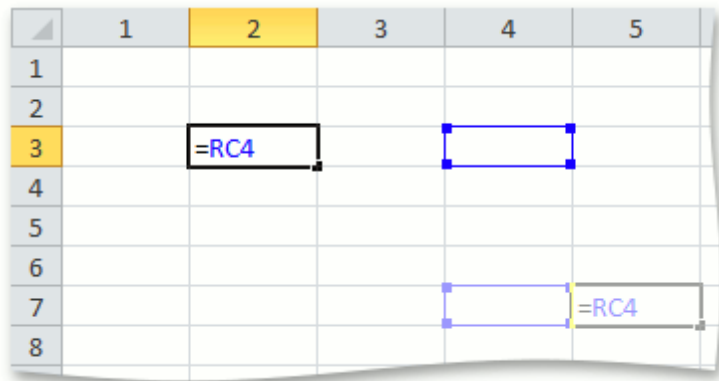
- **Relative Cell References**
- In relative R1C1 cell references, the row and column numbers are enclosed in square brackets. Positive numbers refer to cells below and/or to the right, relative to a cell containing a formula. Negative numbers refer to cells above and/or to the left, relative to the cell containing the formula.
- If an R1C1 cell reference does not include a row or column number, it refers to the cell in the same row or column that contains the cell with the formula.
- When you copy a formula with relative cell references in the R1C1 style, the notation of these cell references remains the same in copied formulas, while the A1 relative references are changed for each copied formula to keep the relative position of the referenced cell and the cell containing the formula.



- **Absolute Cell References**
- In absolute R1C1 cell references, row and column numbers are used without brackets.



- **Mixed Cell References**
- An R1C1 cell reference can either be combined from an absolute reference to a cell row and a relative reference to a cell column (for example, R10C[-5]), or from a relative reference to a cell row and an absolute reference to a cell column (for example, R[3]C5).



Note

If the R1C1 cell reference style is switched on in a workbook, A1 references cannot be used in formulas, and vice-versa. The style of cell references used in formulas is automatically adjusted when you change the active cell reference style in the workbook.

Cross-Worksheet Cell References

In formulas, you can use references to cells located in other worksheets. To do this, specify the worksheet name before the cell reference, and separate them by an exclamation point (!).

When using a worksheet name in a cell reference, enclose it in single quotation marks (') in the following cases.

- The worksheet name contains a character other than a letter or number (spaces, parentheses, braces, etc.).
- The worksheet name starts with neither a letter nor an underscore symbol ("_").
- The worksheet name is the same as an A1 cell reference ("A1", "\$M\$15", etc.).
- The worksheet name starts with an R1C1 cell reference notation.

If you rename a worksheet, this worksheet name is automatically updated in all cell references where it is used.

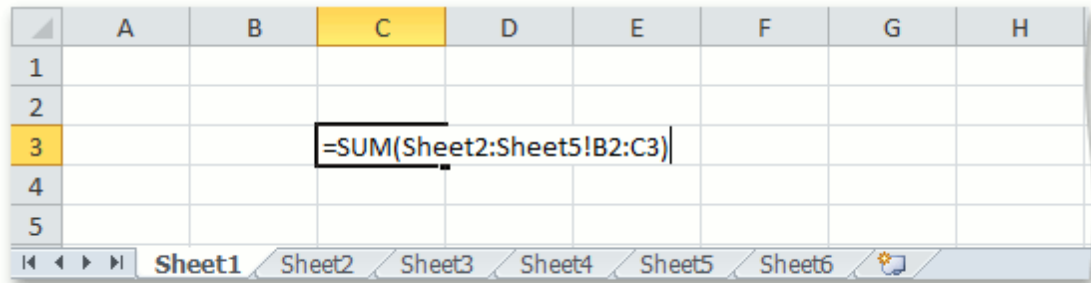
```
C#
// Sum values of cells located in different worksheets.
workbook.Worksheets["Sheet1"].Cells["H15"].Formula = "= Sheet2!C3 + Sheet3!C5";
```

```
Visual Basic
' Sum values of cells located in different worksheets.
workbook.Worksheets("Sheet1").Cells("H15").Formula = "= Sheet2!C3 + Sheet3!C5"
```

3D Cell References


3D references allow you to process data contained in the same cells on multiple worksheets within a workbook. To create a 3D reference, specify the range of worksheet names before the cell (or cell range) reference, and separate them by an exclamation point (!).

For example, the formula in the following image sums all values in cells B2 through C3 located on worksheets Sheet2, Sheet3, Sheet4 and Sheet5. If you insert other worksheets between Sheet2 and Sheet5 (for example, by creating new worksheets, or by duplicating or moving existing worksheets), the B2:C3 cell ranges of the added worksheets will be included into the calculation. If you remove worksheets from the Sheet2:Sheet5 worksheet range, the values of these worksheets are excluded from the calculation.



External Cell References

A reference that refers to a [cell](#), [cell range](#) or [defined name](#) in another workbook is called an *external reference*. In order to use external references, all referenced workbooks should be included in the [Workbook.ExternalWorkbooks](#) collection of the workbook that contains the external references.

 **Note**

Creating external workbooks with circular references between each other is not recommended. In this case, you can get incorrect calculation results, since our internal calculation engine does not trace dependencies between external workbook cells.

An external reference includes the entire path to the workbook file, the workbook file name in square brackets ([]), the worksheet name, an exclamation point (!), and the cell reference. For example, `=c:\Temp\[Book1.xlsx]Sheet1!B3`.

The `ExternalWorkbookCollection.Add` method with the *alias* parameter allows you to specify a custom workbook name that you can use in external references instead of the original file name or when the external workbook is not saved to a file. In this case, an external reference includes the specified workbook name in square brackets ([]), the worksheet name, an exclamation point (!) and the cell reference, as shown in the example below.

C#

```
using DevExpress.Spreadsheet;
// ...
// Access the current workbook where you wish to use external references.
Workbook workbook = new Workbook();
Worksheet worksheet = workbook.Worksheets[0];
// Access the source workbook that contains the cells to be referred to from the current workbook.
Workbook sourceWorkbook = new Workbook();
sourceWorkbook.LoadDocument(@"c:\Temp\Book1.xlsx");
// Add the source workbook to the collection of external workbooks
// and specify a custom name that can be used in external references instead of the file name.
workbook.ExternalWorkbooks.Add(sourceWorkbook, "MyExternalWorkbook");
// Create an external reference in the current workbook.
worksheet["C3"].Formula = "=[MyExternalWorkbook]Sheet1!B3";
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
' Access the current workbook where you wish to use external references.
Dim workbook As New Workbook()
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Access the source workbook that contains the cells to be referred to from the current workbook.
Dim sourceWorkbook As New Workbook()
sourceWorkbook.LoadDocument("c:\Temp\Book1.xlsx")
' Add the source workbook to the collection of external workbooks
' and specify a custom name that can be used in external references instead of the file name.
workbook.ExternalWorkbooks.Add(sourceWorkbook, "MyExternalWorkbook")
' Create an external reference in the current workbook.
worksheet("C3").Formula = "=[MyExternalWorkbook]Sheet1!B3"
```

An external reference to a [defined name](#) in another workbook includes the source workbook name, an exclamation point (!) and the defined name. For example, `=book.xlsx!range_name`.

If the scope of the defined name is a worksheet in another workbook, specify this worksheet name in an external reference. For example, `=book.xlsx]Sheet3!range_name`.

Structured Cell References

Structured references allow you to refer to tables (Table) and different ranges within tables. A structured reference has the following syntax.

`=TableName[[#Data],[ColumnName]]`

- `TableName` is a table name (Table.Name).
- `[#Data]` is a special item specifier that refers to a data range of a table or table column (as in the current example). You can also use the following special item specifiers in structured references, to refer to specific parts of tables or table columns: `[#All]`, `[#Headers]`, `[#Totals]` and `[#This Row]`.
- `[ColumnName]` is a table column specifier (TableColumn.Name). If it is not preceded by any special item specifier, the table column specifier refers to the column data range (excluding the column header and total cells).

To combine column specifiers in structured references, you can use the following operators: colon - to refer to a range of two or more adjacent columns, comma - to refer to a combination of two or more columns, and space - to refer to an intersection of two or more columns.

The image below illustrates examples of structured references using different special item specifiers and reference operators (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F	G	H	I	J
1	Product	Price	Quantity	Discount	Amount					
2	Chocolade	\$5.00	15	3.0%	\$72.75					
3	Konbu	\$9.00	55	10.0%	\$445.50		=Table1[#All],[Amount]			
4	Geitost	\$15.00	70	7.0%	\$976.50					
5				Total:	\$1,494.75					
6										

	A	B	C	D	E	F	G	H	I	J
1	Product	Price	Quantity	Discount	Amount					
2	Chocolade	\$5.00	15	3.0%	\$72.75					
3	Konbu	\$9.00	55	10.0%	\$445.50		=Table1[#Totals]			
4	Geitost	\$15.00	70	7.0%	\$976.50					
5				Total:	\$1,494.75					
6										

	A	B	C	D	E	F	G	H	I	J
1	Product	Price	Quantity	Discount	Amount					
2	Chocolade	\$5.00	15	3.0%	\$72.75					
3	Konbu	\$9.00	55	10.0%	\$445.50		=Table1[Price],Table1[Discount]			
4	Geitost	\$15.00	70	7.0%	\$976.50					
5				Total:	\$1,494.75					
6										

	A	B	C	D	E	F	G	H	I	J
1	Product	Price	Quantity	Discount	Amount					
2	Chocolade	\$5.00	15	3.0%	\$72.75					
3	Konbu	\$9.00	55	10.0%	\$445.50		=Table1[#Headers],[Price]:[Discount]			
4	Geitost	\$15.00	70	7.0%	\$976.50					
5				Total:	\$1,494.75					
6										

See Also
[How to: Use Cell and Worksheet References in Formulas](#)

Formatting Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Cell Basics](#) > [Formatting Cells](#)

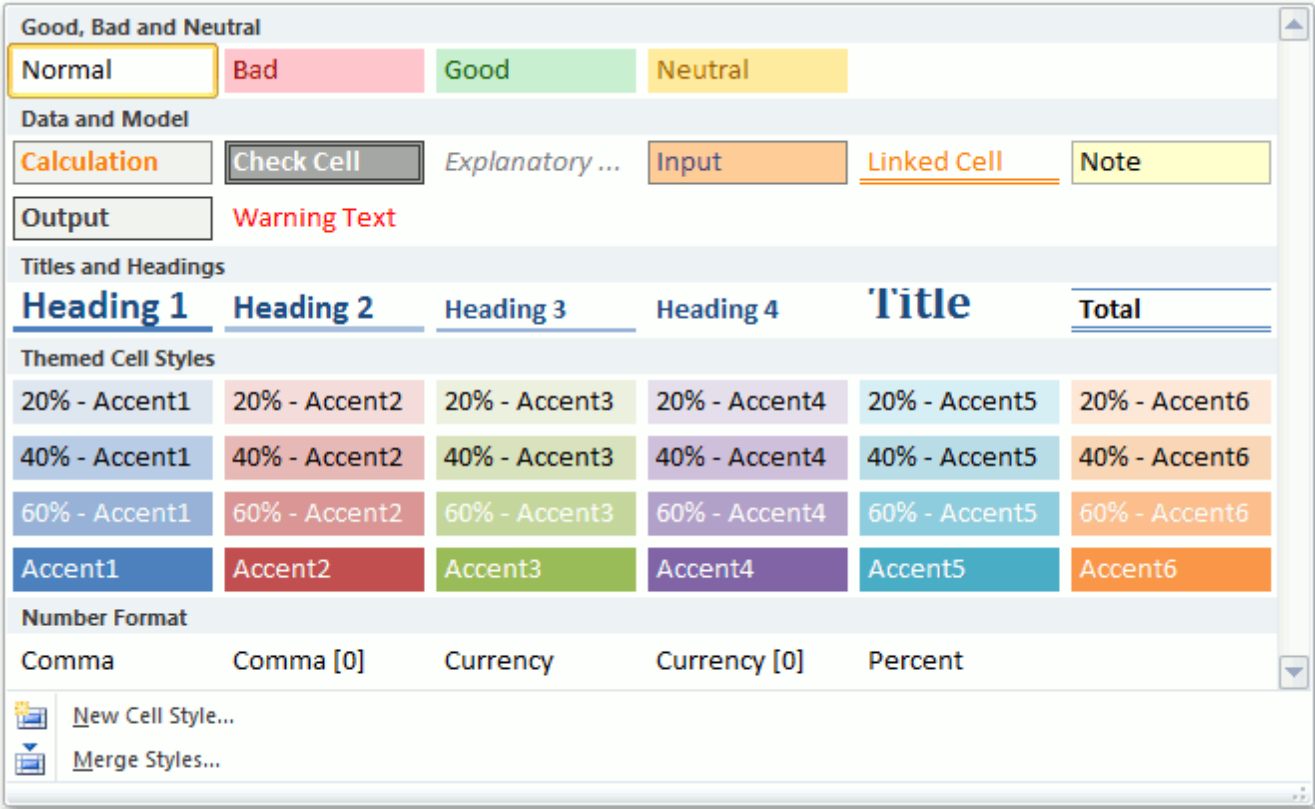
The proper formatting of worksheet [cells](#) improves document appearance and allows end-users to read, find and understand the data that it contains more easily. Cell formatting includes a variety of features such as multiple settings for font, font size, character style (bold, italics, underlined), text alignment, background and foreground colors, etc. This document explains such concepts as [Cell Styles](#), [Direct Cell Formatting](#) and [Style Flags](#), and provides links to [examples](#) on how to format cells using the Spreadsheet Document API.

Cell Styles

A *style* is a named set of predefined cell format characteristics (font settings, number format, content alignment, cell borders, fill color, etc.). When applying a style, all format settings are applied to a cell or cell range in a single step.

A [workbook](#) keeps all available styles in a StyleCollection collection, which is accessed via the [Workbook.Styles](#) property. By default, this collection contains a set of built-in cell styles similar to those found in Microsoft® Excel® (including the *Normal* style that is applied to all unformatted cells in the workbook by default). Identifiers of all built-in styles are listed by the DevExpress.Spreadsheet.BuiltInStyleId enumerator.

The image below shows the gallery of style in Microsoft® Excel®.



You can do the following to manage the workbook's collection of cell styles via the Spreadsheet Document API.

- Modify an existing style by changing the properties of the corresponding Style object. Use the Formatting.BeginUpdate - Formatting.EndUpdate method pair to make multiple modifications to a style.
- Create a new custom style by adding a new Style object to the IWorkbook.Styles collection (the StyleCollection.Add method). Note that new styles are created based on the *Normal* style by default.
- Duplicate an existing style by creating a new style and copying all format settings from an existing style via the Style.CopyFrom method.

For examples, see the [How to: Create or Modify a Style](#) document.

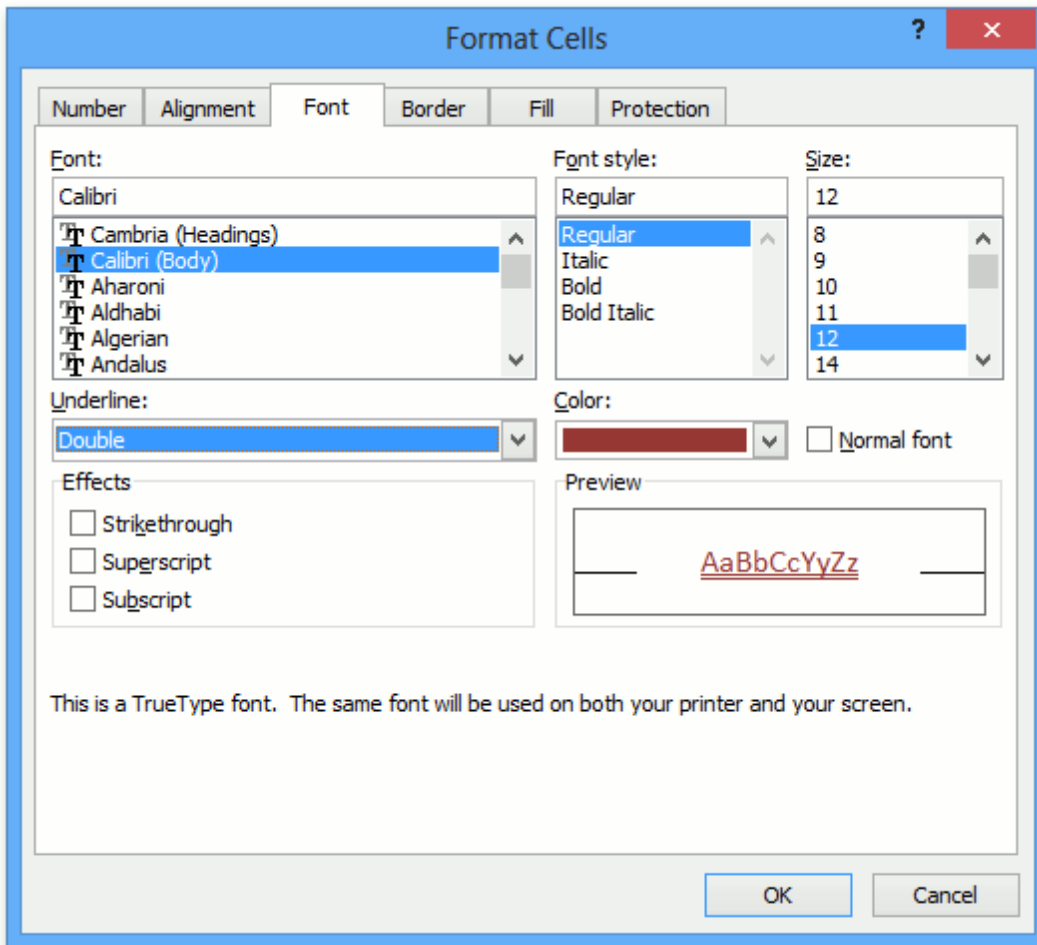
Note	
-------------	--

All custom styles of an Excel document that is loaded into the [Workbook](#) object are automatically added to the [Workbook.Styles](#) collection and can be accessed by their names.

To format a cell or cell range by applying a style, assign the required Style object to the Range.Style property. For details, see the [How to: Apply a Style to a Cell or Range of Cells](#) example.

Direct Cell Formatting

To change cell appearance, you can not only apply a style, but also set the required formatting characteristics for an individual cell or cell range directly. This is called *direct cell formatting*. In Microsoft® Excel®, direct cell formatting options are available via the Ribbon interface or in the **Format Cells** dialog.



To perform direct cell formatting via the Spreadsheet Document API, change the cell or cell range properties that are inherited from the Formatting interface (Formatting.Fill, Formatting.Font, Formatting.Alignment, Formatting.Borders and Formatting.NumberFormat). By default, these properties are set according to the style applied to a cell. Use the following approaches.

- To format an individual cell, access the corresponding Cell object (see [How to: Access a Cell in a Worksheet](#)) and modify its formatting properties.
- To format a range of cells, access and modify the Formatting object using the Range.BeginUpdateFormatting - Range.EndUpdateFormatting method pair.

Thus, a Cell or Range object's properties inherited from the Formatting interface provide access to the actual formatting specified for a cell or range of cells (including the characteristics defined by an applied style and direct formatting attributes).

Style Flags

The actual appearance of a cell is a combination of settings specified by the applied style and the direct cell format settings. Each formatting type provides a set of *flags* (Formatting.Flags). Each flag corresponds to a specific group of format attributes. You can use these flags when formatting a cell, to control whether to use attributes specified in the applied style or attributes specified directly for the cell.

Group	Attributes	Flag
Alignment	Horizontal and vertical alignment of cell content, indentation and text wrap.	StyleFlags.Alignment
Borders	Cell border line styles and colors.	StyleFlags.Borders
Fill	Cell background color.	StyleFlags.Fill
Font	Cell font settings (name, style, color and size).	StyleFlags.Font
Number Format	Cell number format.	StyleFlags.Number
Protection	Cell protection options (Locked and Hidden).	StyleFlags.Protection

Examples

The following examples explain how to format worksheet cells via the Spreadsheet Document API.

- [How to: Format a Cell or Range of Cells](#)
- [How to: Specify Number or Date Format for Cell Content](#)
- [How to: Change Cell Font and Background Color](#)
- [How to: Configure Cell Font Settings](#)
- [How to: Align Cell Content](#)
- [How to: Add and Remove Cell Borders](#)
- [How to: Clear Cell Formatting](#)
- [Conditional Formatting](#)

See Also

[Formatting Cells](#)

Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#)

A spreadsheet formula is an equation that performs a calculation on the numbers, functions and values of one or more cells. A formula is associated with a cell or a cell range. It is accessed using the Range.Formula property.

Formula Types

Formula Type	Description
Shared	A <i>shared formula</i> can be used to optimize calculations and file size. A shared formula is the equivalent of applying the same formula to a range of cells. Shared formula is created automatically without user intervention when you assign a formula string to an array of cells. A specified formula is associated with each cell contained within the specified cell range.
Array	An <i>array formula</i> is a special kind of formula used to perform calculations with arrays of cells. Use the Range.ArrayFormula property or ArrayFormulaCollection.Add methods to include array formulas in a worksheet. An array formula for a cell or a range is accessible via the Range.ArrayFormula property. To find an array formula range that includes a particular cell, use the Cell.GetArrayFormulaRange method. For more information on array formulas, see the Array Formulas document.

Formula Syntax

A formula is a string expression that begins with an equal (=) sign. A formula can contain the constants, operators, cell references, calls to functions, and names.

Consider the following formula, which calculates the mass of a sphere.

=4/3*PI()*(A2^3)*Density

- 4 and 3 are **numeric constants**. Although they are written as integers, the division operator (/) interprets them as being real numbers, i.e., 4.0 and 3.0. As a result, the calculation produces a precise result that is not rounded to an integer.
- / is the division **operator**.
- PI() results in a call to the PI **function**, which returns the value of p.
- A2 is a **cell reference**, which returns the value within that specific cell.
- 3 is a **numeric constant**.
- ^ is the caret **operator**, which raises the left operand to the power of the right operand.
- **Parentheses** are used for grouping and changing the operator precedence.
- * is the asterisk (*) **operator**, which performs multiplication.
- The **Density** is a **defined name** within the worksheet that can represent a cell range, a function or a constant.

The formula is calculated from left to right, according to the operator precedence. To change the order of calculation you can enclose a portion of the formula in parentheses.

Parts of a Formula

- [Operators](#)
- [Functions](#)
- [References](#)
- [Defined Names](#)
- [Array Formulas](#)

Calculation

To recalculate all formulas in a workbook, call the `IWorkbook.Calculate` method. The `DocumentSettings.Calculation` property provides access to calculation options. Calculation results are placed in the `Range.Value` property of corresponding cells.

You can also calculate a formula and leave the document unchanged by using the `IWorkbook.Evaluate` method.

Formula Engine

The `FormulaEngine` is an object that provides the capability to calculate and parse worksheet formulas. It includes a built-in formula parser, as well as the flexibility to evaluate formulas in any range of any worksheet. See the [Formula Engine](#) topic for more information.

See Also

[Cell Data Types](#)
[Formula Examples](#)
[Formula Engine](#)

Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#)

Functions can be used in [formulas](#) to perform simple or complex calculations. You can create a formula with a function that obtains its argument from the result of other functions. A function that is used as an argument is called a nested function. A formula can contain up to seven levels of nested functions.

This document lists the supported functions grouped by the following categories:

- [Mathematical Functions](#)
- [Statistical Functions](#)
- [Date and Time Functions](#)
- [Text Functions](#)
- [Financial Functions](#)
- [Logical Functions](#)
- [Lookup and Reference Functions](#)
- [Engineering Functions](#)
- [Information Functions](#)
- [Compatibility Functions](#)
- [Database Functions](#)
- [Web Functions](#)
- [User-Defined Functions \(UDF\)](#)
- [Real Time Data \(RTD\) function](#)

See Also

[How to: Use Functions and Nested Functions in Formulas](#)

Mathematical Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Mathematical Functions](#)

This document describes the mathematical functions supported by the non-visual spreadsheet component.

Name	Description	Syntax																																										
ABS	Returns the absolute value of a number. The absolute value of a number is the number without its sign.	ABS(number)																																										
ACOS	Returns the arccosine, or inverse cosine, of a number. The returned angle is given in radians in the range of -pi/2 to pi/2.	ACOS(number) The argument must be in the range of -1 to 1.																																										
ACOSH	Returns the inverse hyperbolic cosine of a number.	ACOSH(number)																																										
ACOT	Returns the inverse cotangent (the arccotangent) of a number.	ACOT(number)																																										
ACOTH	Returns the inverse hyperbolic cotangent of the number.	ACOTH(number)																																										
AGGREGATE	Returns an aggregate in a list or database.	<p><i>Reference form:</i> AGGREGATE(function_num, options, ref1, [ref2],) <i>Array form:</i> AGGREGATE(function_num, options, array, [k])</p> <p>The function_num argument specifies the function to use in the calculation. Possible values for the function_num argument are listed in the following table:</p> <table><tr><th>function_num</th><th>Function</th><th>Form</th></tr><tr><td>1</td><td>AVERAGE</td><td>Reference</td></tr><tr><td>2</td><td>COUNT</td><td>Reference</td></tr><tr><td>3</td><td>COUNTA</td><td>Reference</td></tr><tr><td>4</td><td>MAX</td><td>Reference</td></tr><tr><td>5</td><td>MIN</td><td>Reference</td></tr><tr><td>6</td><td>PRODUCT</td><td>Reference</td></tr><tr><td>7</td><td>STDEV.S</td><td>Reference</td></tr><tr><td>8</td><td>STDEV.P</td><td>Reference</td></tr><tr><td>9</td><td>SUM</td><td>Reference</td></tr><tr><td>10</td><td>VAR.S</td><td>Reference</td></tr><tr><td>11</td><td>VAR.P</td><td>Reference</td></tr><tr><td>12</td><td>MEDIAN</td><td>Reference</td></tr><tr><td>13</td><td>MODE.SNGL</td><td>Reference</td></tr></table>	function_num	Function	Form	1	AVERAGE	Reference	2	COUNT	Reference	3	COUNTA	Reference	4	MAX	Reference	5	MIN	Reference	6	PRODUCT	Reference	7	STDEV.S	Reference	8	STDEV.P	Reference	9	SUM	Reference	10	VAR.S	Reference	11	VAR.P	Reference	12	MEDIAN	Reference	13	MODE.SNGL	Reference
function_num	Function	Form																																										
1	AVERAGE	Reference																																										
2	COUNT	Reference																																										
3	COUNTA	Reference																																										
4	MAX	Reference																																										
5	MIN	Reference																																										
6	PRODUCT	Reference																																										
7	STDEV.S	Reference																																										
8	STDEV.P	Reference																																										
9	SUM	Reference																																										
10	VAR.S	Reference																																										
11	VAR.P	Reference																																										
12	MEDIAN	Reference																																										
13	MODE.SNGL	Reference																																										

14	LARGE	Array
15	SMALL	Array
16	PERCENTILE .INC	Array
17	QUARTILE.I NC	Array
18	PERCENTILE .EXC	Array
19	QUARTILE.E XC	Array

The options argument is a number that defines which values to ignore during the calculation.

Options	Behavior
0 or omitted	Ignore nested SUBTOTAL and AGGREGATE functions.
1	Ignore hidden rows, nested SUBTOTAL and AGGREGATE functions.
2	Ignore error values, nested SUBTOTAL and AGGREGATE functions.
3	Ignore hidden rows, error values, nested SUBTOTAL and AGGREGATE functions.
4	Ignore nothing.
5	Ignore hidden rows.
6	Ignore error values.
7	Ignore hidden rows and error values.

Ref1, [ref2], ... specify numeric arguments for the function when the *reference form* is used (you can supply up to 253 ref arguments).

The array argument specifies an array, an array formula, or a reference to a cell range when the *array form* is used. [k] is a second argument required for the following functions: LARGE, SMALL, PERCENTILE.INC,

		QUARTILE.INC, PERCENTILE.EXC, and QUARTILE.EXC.
ARABIC	Converts a Roman numeral to an Arabic numeral.	ARABIC(text)
ASIN	Returns the arcsine, or inverse sine, of a number. The returned angle is given in radians in the range of -pi/2 to pi/2.	ASIN(number) The argument must be in the range of -1 to 1.
ASINH	Returns the inverse hyperbolic sine of a number	ASINH(number)
ATAN	Returns the arctangent, or inverse tangent, of a number. The returned angle is given in radians in the range -pi/2 to pi/2.	ATAN(number)
ATAN2	Calculates the arctangent (i.e. the inverse tangent) of a pair of x and y coordinates, and returns an angle, in radians.	ATAN2(x_num, y_num)
ATANH	Returns the inverse hyperbolic tangent of a given number.	ATANH(number)
BASE	Converts a number into a text representation with the given base.	BASE(number, radix, min_length) The number must be a positive integer less than 2^53. The radix is the base into what the number is converted. The radix is an integer >= 2 and <= 36. Min_length is optional, it is the minimum length of the returned string. If the Min_length parameter is present, leading zeros are added if required.
CEILING	Returns a number rounded up, away from zero, to the nearest multiple of significance.	CEILING(number, significance) The number is the value to round, the significance is the multiple to which you want to round. If the number is negative, and the significance is negative, the value is rounded down, away from zero. If the number is negative, and the significance is positive, the value is rounded up towards zero.
CEILING.MATH	Rounds a number the nearest integer or to the nearest multiple of significance. Regardless of the sign of the number, the number is rounded up.	CEILING.MATH(number, significance, mode) The number is the value to round, the optional significance parameter is the multiple to which you want to round. The optional mode parameter affects negative numbers only and specifies whether the number is rounded toward zero (mode = 0) or away from zero (mode <> 0).
CEILING.PRECISE	Returns number rounded up to the nearest integer or to the nearest multiple of significance.	CEILING.PRECISE(number, significance)
COMBIN	Returns the number of combinations for a given number of items.	COMBIN(n, k) n is the number of items in the set, k is the number of items to choose from the set.
COMBINA	Returns the number of combinations (with repetitions) for a given number of items.	COMBINA(n, k) n is the number of items in the set, k is the number of items to choose from the set.

COS	Returns the cosine of the given angle.	COS(number) The number is the angle in radians.
COSH	Returns the hyperbolic cosine of a number.	COSH(number) The number is any real number.
COT	Returns the cotangent of an angle.	COT(number)
COTH	Returns the hyperbolic cotangent of a number.	COTH(number)
CSC	Returns the cosecant of an angle.	CSC(number)
CSCH	Returns the hyperbolic cosecant of an angle.	CSCH(number)
DECIMAL	Converts a text representation of a number in a given base into a decimal number.	DECIMAL(text, radix)
DEGREES	Converts radians into degrees.	DEGREES(angle) The angle is in radians.
EVEN	Returns a number rounded up to the nearest even integer.	EVEN(number) Regardless of the sign of the number, the value is rounded up when adjusted away from zero. If the number is an even integer, no rounding occurs.
EXP	Returns the value of the mathematical constant e raised to the power of the number.	EXP(number)
FACT	Returns the factorial of a number.	FACT(number) The number is nonnegative. If the number is not an integer, it is truncated.
FACTDOUBLE	Returns the double factorial of a number.	FACTDOUBLE(number)
FLOOR	Rounds the number down toward zero, to the nearest multiple of significance.	FLOOR(number, significance) The number is the value to round, the significance is the multiple to which you want to round.
FLOOR.MATH	Rounds a number down, to the nearest integer or to the nearest multiple of significance.	FLOOR.MATH(number, significance, mode) The number is the value to round, the optional significance is the multiple to which you want to round, the optional mode specifies the direction to round negative numbers (toward 0 if equal to 0, away from 0 if the mode is not zero).
FLOOR.PRECISE	Rounds a number down to the nearest integer or to the nearest multiple of significance. Regardless of the sign of the number, the number is rounded down.	FLOOR.PRECISE(number, significance)
GCD	Returns the greatest common divisor.	GCD(number1, [number2], ...)
INT	Rounds a number down to the nearest integer.	INT(number)
ISO.CEILING	Returns a number that is rounded up to the nearest integer or to the nearest multiple of significance.	ISO.CEILING(number, significance)

		The number is the value to round, the optional significance is the multiple to which you want to round.
LN	Returns the natural logarithm of a number.	LN(number) The number is a positive real number.
LCM	Returns the least common multiple.	LCM(number1, [number2], ...)
LOG	Returns the logarithm of a number to the base that you specify.	LOG(number, base) The number is a positive real number (the base is optional). It is the base of the logarithm. If omitted, the base is assumed to be 10.
LOG10	Returns the base-10 logarithm of a number.	LOG10(number) The number is a positive real number.
MDETERM	Returns the matrix determinant of an array.	MDETERM(array)
MINVERSE	Returns the matrix inverse of an array.	MINVERSE(array)
MMULT	Returns the matrix product of two arrays. The result is an array with the same number of rows as array1 and the same number of columns as array2.	MMULT(array1, array2) The number of columns in array1 must be the same as the number of rows in array2, and both arrays must only contain numbers.
MOD	Returns the remainder after a number is divided by a divisor.	MOD(number, divisor) The number is the number for which to find the remainder, and the divisor is the number by which you want to divide the number. The result has the same sign as the divisor.
MROUND	Returns a number rounded to the desired multiple.	MROUND(number, multiple) The number is the value to round, the multiple is the value to which you want to round the number. The function rounds up, away from zero, if the remainder of dividing the number by multiple is greater than or equal to half the value of the multiple. For example, MROUND(10,3) returns 9 because the remainder of 10 divided by 3 is 1 which is less than half of 3. MROUND(11,3) returns 12.
MUNIT	Returns the unit matrix or the specified dimension.	MUNIT(dimension)
MULTINOMIAL	Returns the multinomial of a set of numbers.	MULTINOMIAL(number1, [number2], ...)
ODD	Rounds a number up to the nearest odd integer.	ODD(number)
PI	Returns the number 3.14159265358979, the mathematical constant pi, accurate to 15 digits.	PI()
POWER	Returns the result of a number raised to a power.	POWER(number, power) The number is a real number, and the power is the exponent to which the number is raised. You can use the "^" operator instead.
PRODUCT	Multiplies all numbers given as arguments and returns the product.	PRODUCT(number1, number2, ...)

		If an argument is an array or reference, only numbers in the array or reference are multiplied. Empty cells, logical values, and text in the array or reference are ignored.
QUOTIENT	Returns the integer portion of a division.	QUOTIENT(number, divisor)
RADIANS	Converts degrees to radians.	RADIANS(angle)
RAND	Returns an evenly distributed random real number greater than or equal to 0 and less than 1. A new random real number is returned every time the worksheet is calculated.	RAND()
RANDBETWEEN	Returns a random integer number between the numbers that are specified. A new random integer number is returned every time the worksheet is calculated.	RANDBETWEEN(bottom, top) The bottom is the smallest integer, the top is the largest integer that the function will return.
ROMAN	Converts an arabic numeral to roman, as text.	ROMAN(number, [form])
ROUND	Rounds a number to a specified number of digits.	ROUND(number, num_digits) If num_digits is 0, the number is rounded to the nearest integer. If num_digits is greater than 0 (zero), then the number is rounded to the specified number of decimal places. If num_digits is less than 0, then the number is rounded to the left of the decimal point.
ROUNDDOWN	Rounds a number down, toward zero, to a specified number of digits.	ROUNDDOWN(number, num_digits) If num_digits is 0, the number is rounded to the nearest integer. If num_digits is greater than 0 (zero), then the number is rounded to the specified number of decimal places. If num_digits is less than 0, then the number is rounded to the left of the decimal point.
ROUNDUP	Rounds a number up to a specified number of digits.	ROUNDUP(number, num_digits) If num_digits is 0, the number is rounded to the nearest integer. If num_digits is greater than 0 (zero), then the number is rounded to the specified number of decimal places. If num_digits is less than 0, then the number is rounded to the left of the decimal point.
SEC	Returns the secant of an angle.	SEC(number)
SECH	Returns the hyperbolic secant of an angle.	SECH(number)
SERIESSUM	Returns the sum of a power series based on the formula.	SERIESSUM(x, n, m, coefficients)
SIGN	Determines the sign of a number. Returns 1 if the number is positive, zero (0) if the number is 0, and -1 if the number is negative.	SIGN(number)
SIN	Returns the sine of the given angle.	SIN(number) The number is the angle in radians.
SINH	Returns the hyperbolic sine of a number.	SIN() number

SQRT	Returns a positive square root.	SQRT(number) The number is any positive number.																																				
SQRTPI	Multiplies a specified number by pi and returns a square root of the product.	SQRTPI(number) The number is any positive number.																																				
SUBTOTAL	Performs a specified calculation (the sum, product, average, etc.) for a supplied set of values	<p>SUBTOTAL(function_num, ref1, ref2, ...) The function_num argument is a number that specifies the calculation type. Possible values for the function_num argument are listed in the following table. Note that by selecting a proper function_num argument you can ignore or include hidden values in calculation.</p> <table><tr><th>function_num (include hidden values)</th><th>function_num (ignore hidden values)</th><th>Function</th></tr><tr><td>1</td><td>101</td><td>AVERAGE</td></tr><tr><td>2</td><td>102</td><td>COUNT</td></tr><tr><td>3</td><td>103</td><td>COUNTA</td></tr><tr><td>4</td><td>104</td><td>MAX</td></tr><tr><td>5</td><td>105</td><td>MIN</td></tr><tr><td>6</td><td>106</td><td>PRODUCT</td></tr><tr><td>7</td><td>107</td><td>STDEV</td></tr><tr><td>8</td><td>108</td><td>STDEVP</td></tr><tr><td>9</td><td>109</td><td>SUM</td></tr><tr><td>10</td><td>110</td><td>VAR</td></tr><tr><td>11</td><td>111</td><td>VARP</td></tr></table>	function_num (include hidden values)	function_num (ignore hidden values)	Function	1	101	AVERAGE	2	102	COUNT	3	103	COUNTA	4	104	MAX	5	105	MIN	6	106	PRODUCT	7	107	STDEV	8	108	STDEVP	9	109	SUM	10	110	VAR	11	111	VARP
function_num (include hidden values)	function_num (ignore hidden values)	Function																																				
1	101	AVERAGE																																				
2	102	COUNT																																				
3	103	COUNTA																																				
4	104	MAX																																				
5	105	MIN																																				
6	106	PRODUCT																																				
7	107	STDEV																																				
8	108	STDEVP																																				
9	109	SUM																																				
10	110	VAR																																				
11	111	VARP																																				
SUM	Adds all numbers that you specify as arguments	SUM(number1, number2, ...) Each argument can be a range, a cell reference, an array, a constant, a formula, or the result from another function. If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, or text in the array or reference are ignored.																																				
SUMIF	Sums the values in a range that meet criteria that you specify	SUMIF(range, criteria, sum_range) Range is the range of cells that you wish to filter by criteria. Criteria is a number, expression, cell reference, text, or function that defines which cells will be added. Optional sum_range specifies the actual cells to add. If omitted, the cells specified in the Range argument are added.																																				
SUMIFS	Sums the cells in a range that meet multiple criteria.	SUMIFS(sum_range, criteria_range1, criteria1, criteria_range2, criteria2, ...) Sum_range is the range of cells to sum. Criteria_range specifies the range in which to evaluate the specified criteria. The criteria is a number, expression, cell reference or text																																				

		that defines which cells in the criteria_range argument will be added. You can specify several criteria ranges with specific criteria for each.
SUMPRODUCT	Returns the sum of the products of the corresponding values in two or more supplied arrays.	SUMPRODUCT(array1, array2, array3, ...) The array arguments must have the same dimensions. Array entries that are not numeric are treated as if they were zeros.
SUMSQ	Returns the sum of the squares of the arguments.	SUMSQ(number1, number2, ...) You can also use a single array or a reference to an array instead of arguments separated by commas. Empty cells, logical values, text, or error values in the array or reference are ignored.
SUMX2MY2	Returns the sum of the difference of squares of corresponding values in two arrays.	SUMX2MY2(array_x, array_y)
SUMX2PY2	Returns the sum of the sum of squares of corresponding values in two arrays.	SUMX2PY2(array_x, array_y)
SUMXMY2	Returns the sum of squares of differences of corresponding values in two arrays	SUMXMY2(array_x, array_y) Array_x and array_y should have the same number of values. If an array or reference argument contains text, logical values, or empty cells, those values are ignored.
TAN	Returns the tangent of the given angle.	TAN(angle) The angle is measured in radians.
TANH	Returns the hyperbolic tangent of a number.	TANH(number) The number is any real number.
TRUNC	Truncates a number to a specified number of decimal places.	TRUNC(number, num_digits) If an optional parameter num_digits is omitted or has a value of 0, rounding is performed on an integer.

See Also

[How to: Use Functions and Nested Functions in Formulas](#)

Statistical Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Statistical Functions](#)

This document describes statistical functions supported by a non-visual spreadsheet component.

***Standard Options** means the following:

The arguments 'number1', '[number2]', etc., are numerical values or references to cells containing numbers. You can enter up to 255 number arguments. The arguments can be logical values or text representations of numbers. You can also use a reference to a cell array instead of separate numbers. Text, logical values, or empty cells within an array are ignored.

Name	Description	Syntax
AVEDEV	Calculates the average deviation of a set of values.	AVEDEV(number1, [number2], ...) *Standard Options
AVERAGE	Returns the average (arithmetic mean) of a list of numbers.	AVERAGE(number1, [number2], ...) *Standard Options
AVERAGEA	Returns the average (arithmetic mean) of a list of numbers.	AVERAGEA(number1, [number2], ...) The difference compared to the AVERAGE function is that within arrays or reference arguments logical values are counted as 1 or 0 and text is counted as zero.
AVERAGEIF	Returns the average (arithmetic mean) of the cells in a range that meet a given criteria.	AVERAGEIF(range, criteria, [average_range]) The range is an array of values (or range of cells containing values) that is tested against the given criteria. If the average_range argument is omitted, the values in the initial range argument are used to calculate an average. Otherwise, the corresponding cells in the average_range array is used for calculation.
AVERAGEIFS	Finds entries in one or more arrays, that satisfy the respective supplied criteria, and returns the average (arithmetic mean) of the corresponding values in an array supplied as the first argument.	AVERAGEIFS(average_range, criteria_range1, criteria1, [criteria_range2, criteria2], ...) The average_range is the range for calculation. Criteria_range1, [criteria_range2], - are the arrays to be tested against supplied criteria. Criteria1, [criteria2], - are the respective conditions to be tested.
BETA.INV	Returns the inverse of the beta cumulative probability density function (BETA.DIST).	BETA.INV(probability,alpha,beta,[A],[B])
BETA.DIST	Returns the beta distribution.	BETA.DIST(x,alpha,beta,cumulative,[A],[B])
BINOM.DIST	Returns the individual term binomial distribution probability.	BINOM.DIST(number_s,trials,probability_s,cumulative)
BINOM.INV	Returns the smallest value for which the cumulative binomial distribution is greater than or equal to a criterion value.	BINOM.INV(trials,probability_s,alpha)
BINOM.DIST.RANGE	Returns the probability of a trial result using a binomial distribution.	BINOM.DIST.RANGE(trials,probability_s,number_s,[number_s2])

CHISQ.DIST	Returns the chi-squared distribution.	CHISQ.DIST(x,deg_freedom,cumulative) X - Required. The value at which you want to evaluate the distribution. Deg_freedom - Required. The number of degrees of freedom. Cumulative - Required. A logical value that determines the form of the function.
CHISQ.DIST.RT	Returns the right-tailed probability of the chi-squared distribution.	CHISQ.DIST.RT(x,deg_freedom)
CHISQ.INV	Returns the inverse of the left-tailed probability of the chi-squared distribution.	CHISQ.INV(probability,deg_freedom)
CHISQ.INV.RT	Returns the inverse of the right-tailed probability of the chi-squared distribution.	CHISQ.INV.RT(probability,deg_freedom)
CHISQ.TEST	Returns the test for independence as the value from the chi-squared distribution for the statistic and the appropriate degrees of freedom.	CHISQ.TEST(actual_range,expected_range)
CONFIDENCE.NORM	Returns the confidence interval for a population mean, using a normal distribution.	CONFIDENCE.NORM(alpha,standard_dev,size) Alpha is the significance level used to compute the confidence level. The confidence level equals 100*(1 - alpha)%, or in other words, an alpha of 0.05 indicates a 95 percent confidence level. Standard_dev is the population standard deviation for the data range and is assumed to be known. Size is the sample size.
CONFIDENCE.T	Returns the confidence interval for a population mean, using a Student's t distribution.	CONFIDENCE.T(alpha,standard_dev,size)
CORREL	Calculates the correlation coefficient for two sets of values.	CORREL(array1, array2) The arrays should be of equal length.
COUNT	Returns the number of numeric values in a set of cells or values.	COUNT(value1, [value2], ...) Numbers and dates are always counted as numeric values. Text representations of numbers and logical values are counted only if supplied directly as function arguments.
COUNTA	Returns a number of non-blank cells or values within a specified set.	COUNTA(value1, [value2], ...)
COUNTBLANK	Returns the number of blank cells in a range of cells.	COUNTBLANK(range)
COUNTIF	Returns the number of cells that satisfy a given criteria in a specified range.	COUNTIF(range, criteria)
COUNTIFS	Returns the number of entries that satisfy all specified criteria in specified ranges.	COUNTIFS(criteria_range1, criteria1, [criteria_range2, criteria2], ...) Each additional range must have the same number of rows and columns as the criteria_range1 argument. The ranges do not have to be adjacent to each other. Criteria is applied to the associated range and the logical matrix (true/false) is calculated. Resulting

		matrices are added using AND operator and the number of True entries is counted.
COVARIANCE.P	Returns population covariance, the average of the products of deviations for each data point pair in two data sets.	COVARIANCE.P(array1,array2)
COVARIANCE.S	Returns the sample covariance, the average of the products of deviations for each data point pair in two data sets.	COVARIANCE.S(array1,array2)
DEVSQ	Returns the sum of squares of deviations of data points from their sample mean.	DEVSQ(number1, [number2], ...)
EXPON.DIST	Returns the exponential distribution.	EXPON.DIST(x,lambda,cumulative)
F.DIST	Returns the F probability distribution.	F.DIST(x,deg_freedom1,deg_freedom2,cumulative)
F.DIST.RT	Returns the (right-tailed) F probability distribution (degree of diversity) for two data sets.	F.DIST.RT(x,deg_freedom1,deg_freedom2)
F.INV	Returns the inverse of the F probability distribution.	F.INV(probability,deg_freedom1,deg_freedom2)
F.INV.RT	Returns the inverse of the (right-tailed) F probability distribution.	F.INV.RT(probability,deg_freedom1,deg_freedom2)
F.TEST	Returns the result of an F-test, the two-tailed probability that the variances in array1 and array2 are not significantly different.	F.TEST(array1,array2)
FISHER	Returns the Fisher transformation at x.	FISHER(x)
FISHERINV	Returns the inverse of the Fisher transformation.	FISHERINV(y)
FORECAST	Performs a linear regression to fit a straight line using least squares criterion to the specified arrays of values. Returns a new value for the specified X data point.	FORECAST(x,known_y's,known_x's) X is the data point for which you want to predict a value. Known_y's is the set of y-values for the relationship $y = mx + b$. Known_x's is a set of x-values for the relationship.
FREQUENCY	Calculates how often values occur within a range of values, and then returns a vertical array of numbers.	FREQUENCY(data_array, bins_array) Data_array is an array of or reference to a set of values for which you want to count frequencies. If data_array contains no values, FREQUENCY returns an array of zeros. Bins_array is an array of or reference to intervals into which you want to group the values in data_array. If bins_array contains no values, FREQUENCY returns the number of elements in data_array.
GAMMA	Return the gamma function value.	GAMMA(number)
GAMMA.DIST	Returns the gamma distribution.	GAMMA.DIST(x,alpha,beta,cumulative)

GAMMA.INV	Returns the inverse of the gamma cumulative distribution.	GAMMA.INV(probability,alpha,beta)
GAMMALN	Returns the natural logarithm of the gamma function.	GAMMALN(x)
GAMMALN.PRECISE	Returns the natural logarithm of the gamma function (new version).	GAMMALN.PRECISE(x)
GAUSS	Calculates the probability that a member of a standard normal population will fall between the mean and z standard deviations from the mean.	GAUSS(z)
GEOMEAN	Returns the geometric mean of a list of numbers.	GEOMEAN(number1, [number2], ...) *Standard Options
GROWTH	Calculates an exponential curve that best fits your data based on a number of known X and Y values. Returns the y-values for a series of new x-values.	GROWTH(known_y's, [known_x's], [new_x's], [const]) Known_y's is the set of y-values for the relationship $y = b * m^x$. Known_x's is an optional set of x-values for the relationship; if omitted, it is assumed to be the array {1,2,3,...} that is the same size as known_y's. New_x's is an optional set of new x-values for which you want GROWTH to return corresponding y-values. If omitted, it is assumed to be the same as known_x's. Const is an optional logical value. It is TRUE or omitted, to calculate the b constant; otherwise b is set to 1. GROWTH is the exponential counterpart to the linear regression function TREND.
HARMEAN	Returns the harmonic mean of a data set.	HARMEAN(number1,number2,...)
HYPGEOM.DIST	Returns the hypergeometric distribution.	HYPGEOM.DIST(sample_s,number_sample,population_s,number_pop,cumulative)
INTERCEPT	Calculates the best fit regression line using a series of x- and y- values and returns the value at which this line intercepts the y-axis.	INTERCEPT(known_y's, known_x's) Known_y's is the dependent set of observations or data. Known_x's is the independent set of observations or data.
KURT	Returns the kurtosis of a data set.	KURT(number1,number2,...)
LARGE	Returns the k'th largest value from an array or a range of cells containing numerical values.	LARGE(array, k) The array argument is the array or range of data for which the k-th largest value will be determined. The k argument is the top position of value in a sorted array.
LINEST	Returns statistical information on the line of best fit, through a supplied set of x- and y- values using 'least-square' method.	LINEST(known_y's,known_x's,const,stats) Known_y's is the set of y-values you already know in the relationship $y = mx + b$. Known_x's is an optional set of x-values that you may already know in the relationship $y = mx + b$. Const is a logical value specifying whether to force the constant b to equal 0. Stats is

		a logical value specifying whether to return additional regression statistics.
LOGEST	Calculates regression to fit an exponential curve using a least squares method.	LOGEST(known_y's, [known_x's], [const], [stats] Known_y's is the set of dependent values. Known_x's is an optional set of independent values. The argument ""constant"" is TRUE to calculate the constant b in the regression equation $y = b * m^x$; otherwise, b equals 1. The argument ""stats"" set to TRUE if you want additional statistics, including various sums of squares, r-squared, f-statistic, and standard errors of the regression coefficients. LOGEST is the exponential counterpart to the linear regression function LINEST.
LOGNORM.DIST	Returns the log-normal probability density function or the cumulative log-normal distribution.	LOGNORM.DIST(x,mean,standard_dev, cumulative) X is the value at which to evaluate the function. Mean is the mean of $\ln(x)$. Standard_dev is the standard deviation of $\ln(x)$. Cumulative is a logical value that determines the form of the function. If cumulative is TRUE, LOGNORM.DIST returns the cumulative distribution function; otherwise, it returns the probability density function.
LOGNORM.INV	Returns the inverse of the log-normal distribution	LOGNORM.INV(probability,mean,standard_dev) X is a probability associated with the lognormal distribution. Mean is the mean of $\ln(x)$. Standard_dev is the standard deviation of $\ln(x)$.
MAX	Returns the largest value in a set of values.	MAX(number1, [number2], ...) *Standard Options
MAXA	Returns the largest value in a set of values.	MAXA(number1, [number2], ...) The difference compared to the MAX function is that within arrays or reference arguments logical values are counted as 1 or 0 and text is counted as zero.
MEDIAN	Returns the statistical median (the middle value) of a list of numbers.	MEDIAN(number1, [number2], ...) *Standard Options
MIN	Returns the smallest value in a set of values.	MIN(number1, [number2], ...) *Standard Options
MINA	Returns the smallest value in a set of values.	MIN(number1, [number2], ...) The difference compared to the MIN function is that within arrays or reference arguments logical values are counted as 1 or 0 and text is counted as zero.
MODE.MULT	Returns a vertical array of the statistical modes (the most frequently occurring values) within a set of values.	MODE.MULT((number1,[number2],...]) The arguments number1, [number2], etc, are numerical values or references to cells containing numbers. You can enter up to 255 number arguments. The arguments can be logical values or text representations of numbers. You can also use a reference to cell array

		instead of separate numbers. Text, logical values, or empty cells within array are ignored. If the data set contains no duplicate data points, the #N/A error value is returned.
MODE.SNGL	Returns the statistical mode (the most frequently occurring value) in a set of values.	MODE.SNGL(number1,[number2],...) The arguments number1, [number2], etc, are numerical values or references to cells containing numbers. You can enter up to 255 number arguments. The arguments can be logical values or text representations of numbers. You can also use a reference to cell array instead of separate numbers. Text, logical values, or empty cells within array are ignored. If the data set contains no duplicate data points, the #N/A error value is returned.
NEGBINOM.DIST	Returns the negative binomial distribution.	NEGBINOM.DIST(number_f,number_s, probability_s,cumulative)
NORM.DIST	Returns the normal distribution for the specified mean and standard deviation.	NORM.DIST(x,mean,standard_dev,cumulative) X is the value for which you want the distribution. Mean is the arithmetic mean of the distribution. Standard_dev is the standard deviation of the distribution. Cumulative is a logical value that determines the form of the function. If cumulative is TRUE, NORM.DIST returns the cumulative distribution function; if FALSE, it returns the probability mass function.
NORM.INV	Returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.	NORM.INV(probability,mean,standard_dev) Probability is a probability corresponding to the normal distribution. Mean is the arithmetic mean of the distribution. Standard_dev is the standard deviation of the distribution.
NORM.S.DIST	Returns the standard normal distribution (has a mean of zero and a standard deviation of one).	NORM.S.DIST(z,cumulative) Z is the value for which you want the distribution. Cumulative is a logical value that determines the form of the function. If cumulative is TRUE, NORMS.DIST returns the cumulative distribution function; otherwise it returns the probability mass function.
NORM.S.INV	Returns the inverse of the standard normal cumulative distribution.	NORM.S.INV(probability) Probability is a probability that corresponds to the normal distribution.
PEARSON	Returns the Pearson product moment correlation coefficient, a statistical measurement of the correlation (linear association) between two sets of values.	PEARSON(array1,array2) The array1 is a set of independent values, the array2 is a set of dependent values.
PERCENTILE.EXC	Returns the k'th percentile of a supplied range of values for a given value of k, within the range 0 to 1 (exclusive).	PERCENTILE.EXC(array, k)

PERCENTILE.INC	Returns the k'th percentile of a supplied range of values for a given value of k, within the range 0 to 1 (inclusive).	PERCENTILE.INC(array, k)
PERCENTRANK.EXC	Calculates the relative position, between 0 and 1 (exclusive), of a specified value within a supplied array.	PERCENTRANK.EXC(array, x, [significance])
PERCENTRANK.INC	Calculates the relative position, between 0 and 1 (inclusive), of a specified value within a supplied array.	PERCENTRANK.INC(array, x, [significance])
PERMUT	Returns the number of permutations for a given number of objects.	PERMUT(number, number_chosen)
PERMUTATIONA	Returns the number of permutations for a given number of objects (with repetitions) that can be selected from the total objects.	PERMUTATIONA(number, number_chosen)
PHI	Returns the value of the density function for a standard normal distribution.	PHI(x)
POISSON.DIST	Returns the Poisson distribution.	POISSON.DIST(x,mean,cumulative)
PROB	Returns the probability that values in a range are between two limits.	PROB(x_range, prob_range, [lower_limit], [upper_limit])
QUARTILE.EXC	Returns the quartile of the data set, based on percentile values from 0..1, exclusive.	QUARTILE.EXC(array, quart)
QUARTILE.INC	Returns the quartile of a data set, based on percentile values from 0..1, inclusive.	QUARTILE.INC(array,quart)
RANK.AVG	Returns the rank of a number in a list of numbers: its size relative to other values in the list; if more than one value has the same rank, the average rank is returned.	RANK.AVG(number,ref,[order])
RANK.EQ	Returns the rank of a number in a list of numbers. Its size is relative to other values in the list; if more than one value has the same rank, the top rank of that set of values is returned.	RANK.EQ(number,ref,[order])
RSQ	Returns the square of the Pearson product moment correlation coefficient.	RSQ(known_y's,known_x's)
SKEW	Returns the skewness (the asymmetry around the mean) of a distribution.	SKEW(number1, [number2], ...) Number1 is required, subsequent numbers are optional. You can supply up to 255 arguments for which you want to calculate skewness or use a single array or a reference to an array instead of arguments separated by commas.
SKEW.P	Returns the skewness of a distribution based on a population: a characterization of the degree of asymmetry of a distribution around its mean.	SKEW.P(number 1, [number 2],)

SLOPE	Returns the slope of the linear regression line through data points in known_y's and known_x's.	SLOPE(known_y's, known_x's) Known_y's is an array or cell range of numeric dependent data values. Known_x's is the set of independent data values.
SMALL	Returns the k'th smallest value from an array or a range of cells containing numerical values.	SMALL array, k) The array argument is the array or range of data for which the k-th smallest value will be determined. The k argument is the top position of value in an array sorted from smallest to largest.
STANDARDIZE	Returns a normalized value from a distribution characterized by mean and standard_dev.	STANDARDIZE(x,mean,standard_dev)
STEYX	Returns the standard error of the predicted y-value for each x in the regression.	STEYX(known_y's, known_x's)
STDEVA	Calculates the standard deviation based on a sample.	STDEVA(value1, [value2], ...) The arguments value1, [value2], etc, are numerical values or references to cells. You can enter up to 255 arguments. The arguments can be logical values or text representations of numbers. You can also use a reference to cell array instead of separate numbers. Arguments that contain TRUE evaluate as 1; arguments that contain text or FALSE evaluate as 0 (zero).
STDEVPA	Calculates standard deviation based on the entire population, including numbers, text, and logical values	STDEVPA(value1,value2,...)
STDEV.P	Calculates the standard deviation based on the entire population.	STDEV.P(number1,[number2],...) *Standard Options
STDEV.S	Calculates the standard deviation based on a sample.	STDEV.S(number1,[number2],...) *Standard Options
T.DIST	Returns the Student's t-distribution.	T.DIST(x,deg_freedom, cumulative)
T.DIST.2T	Returns the two-tailed Student's t-distribution.	T.DIST.2T(x,deg_freedom)
T.DIST.RT	Returns the right-tailed Student's t-distribution.	T.DIST.RT(x,deg_freedom)
T.INV	Returns the left-tailed inverse of the Student's t-distribution.	T.INV(probability,deg_freedom)
T.TEST	Returns the probability that is associated with a Student's t-Test.	T.TEST(array1,array2,tails,type)
T.INV.2T	Returns the two-tailed inverse of the Student's t-distribution.	T.INV.2T(probability,deg_freedom)
TREND	Performs a linear regression to fit a straight line using least squares criterion to the specified arrays of values.	TREND(known_y's, [known_x's], [new_x's], [const]) Known_y's is the set of y-values for the relationship $y = mx + b$. Known_x's is an optional set of x-values for the relationship; if omitted, it is assumed to be the array {1,2,3,...} that is the

		same size as known_y's. New_x's is an optional set of new x-values for which you want TREND to return corresponding y-values. If omitted, it is assumed to be the same as known_x's. Const is an optional logical value. It is TRUE or omitted, to calculate the b constant; otherwise b is set to 0 (zero).
TRIMMEAN	Returns the mean of the interior of a data set.	TRIMMEAN(array, percent)
VARA	Estimates variance based on a sample, including numbers, text, and logical values.	VARA(value1, [value2], ...)
VARPA	Calculates variance based on the entire population, including numbers, text, and logical values.	VARPA(value1, [value2], ...)
VAR.P	Calculates variance for the entire population.	VAR.P(number1,[number2],...] *Standard Options
VAR.S	Calculates variance for the sample.	VAR.S(number1,[number2],...] *Standard Options
WEIBULL.DIST	Returns the Weibull distribution.	WEIBULL.DIST(x,alpha,beta,cumulative)
Z.TEST	Returns the one-tailed probability-value of a z-test.	Z.TEST(array,x,[sigma])

See Also

[How to: Use Functions and Nested Functions in Formulas](#)

Date and Time Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Date and Time Functions](#)

This document describes the date and time functions supported by the non-visual spreadsheet component.

Name	Description	Syntax
DATE	Returns a sequential serial number that represents a particular date.	<p>DATE(year,month,day)</p> <p>Year - A number that may include from one to four digits. If the year is between 0 (zero) and 1899 (inclusive), this value is added to 1900 to calculate the year. If the year is between 1900 and 9999 (inclusive), this value is used as the year. If the year is less than 0 or is 10000 or greater, the #NUM! error is returned.</p> <p>Month - A positive or negative integer representing the month of the year. If the month is greater than 12, this number is added to the first month in the year specified. If the month is less than 1, this value plus 1 is subtracted from the first month in the year specified.</p> <p>Day - A positive or negative integer representing the day of the month. If the day is greater than the number of days in the month specified, this number of days is added to the first day in the month. If the day is less than 1, this value plus 1 is subtracted from the first day of the month specified.</p>
DATEDIF	Returns the number of days, months, or years between two dates.	<p>DATEDIF(start_date, end_date, interval_type)</p> <p>Start_date is the first date of the period.</p> <p>End_date is the last date of the period.</p> <p>Dates can be entered as text strings within quotation marks, as serial numbers, or as the results of other formulas or functions.</p> <p>Interval_type is a case-insensitive string that specifies the type of the returned information:</p> <ul style="list-style-type: none"> • "M" - Complete calendar months between dates. • "D" - Number of days between dates. • "Y" - Complete calendar years between dates. • "YM" - Complete calendar months between dates as if they were of the same year. • "YD" - Complete calendar days between dates as if they were of the same year. • "MD" - Complete calendar days between dates as if they were of the same month and year.
DATEVALUE	Converts a date's text representation into a serial number that represents a date.	DATEVALUE(date_text)
DAY	Returns an integer representing the day of the month (from 1 - 31).	<p>DAY(date_serial_number)</p> <p>Date_serial_number is a serial number representing a date.</p>

		Use functions such as DATE to supply an argument to the DAY function.
DAYS	Returns the number of days between two dates.	DAYS(end_date, start_date)
DAYS360	Returns the number of days between two dates, based on a 360-day year (twelve 30-day months).	DAYS360(start_date,end_date,[method]) Start_date and end_date are two dates between which you want to know the number of days. Dates should be supplied as cell references, or as results of the DATE or other function that returns a sequential serial number representing a particular date. Method is a logical value that specifies whether to use the US (NASD) method (if set to True) or European method (if set to False) in the calculation. If the method argument is omitted, the NASD method is used.
EDATE	Returns a date that is a specified number of months before or after a start date.	EDATE(start_date, months) If months is not an integer, it is truncated. Dates should be supplied as cell references, or as results of the DATE or other function that returns a sequential serial number representing a particular date.
EOMONTH	Returns the last day of the month that is several months ahead or prior to the start_date.	EOMONTH(start_date, months) If months is not an integer, it is truncated. Dates should be supplied as cell references or as results of the DATE or other function that returns a sequential serial number representing a particular date.
HOUR	Returns an integer that is the hour component of a specified time.	HOUR(time) Time can be a text string representing time, the result of other functions (such as TIMEVALUE), or a decimal number. The decimal number is calculated as a portion of a date value (24 hours is 1.0, so 3 AM is 0.125 in decimal time).
ISOWEEKNUM	Returns number of the ISO week number of the year for a given date.	ISOWEEKNUM(date)
MINUTE	Returns an integer that is the minutes component of a specified time.	MINUTE(time) Time can be a text string representing time, the result of other functions (such as TIMEVALUE), or a decimal number. The decimal number is calculated as a portion of a date value (24 hours is 1.0, so 3 AM is 0.125 in decimal time).
MONTH	Returns an integer that is the month component of a specified date.	MONTH(date_serial_number) Date_serial_number is a serial number representing a date. Use functions such as DATE to supply an argument to the MONTH function.
NETWORKDAYS	Calculates the number of work days between two dates.	NETWORKDAYS(start_date,end_date,holidays) Dates should be supplied as cell references or as results of the DATE or other function that returns a sequential serial number

		representing a particular date. All weekdays are included; weekends and dates identified as holidays are excluded, and not counted as working days. To specify the holidays to be excluded, use an optional third argument that is a range of cells containing dates or an array of serial numbers representing dates.																														
NETWORKDAYS.INTL	Returns the number of whole workdays between two dates using parameters to indicate which and how many days are weekend days.	<p>NETWORKDAYS.INTL(start_date, end_date, weekend, holidays)</p> <p>Start_date and end_date are required. They are the dates for which the difference is to be computed.</p> <p>Weekend parameter is optional. It indicates the days of the week that are weekend days and are not included in the number of whole working days. Weekend is a weekend number or string that specifies when weekends occur. Weekend numbers are listed in the following table.</p> <table><tr><th>Weekend Number</th><th>Weekend Days</th></tr><tr><td>1 or omitted</td><td>Saturday, Sunday</td></tr><tr><td>2</td><td>Sunday, Monday</td></tr><tr><td>3</td><td>Monday, Tuesday</td></tr><tr><td>4</td><td>Tuesday, Wednesday</td></tr><tr><td>5</td><td>Wednesday, Thursday</td></tr><tr><td>6</td><td>Thursday, Friday</td></tr><tr><td>7</td><td>Friday, Saturday</td></tr><tr><td>11</td><td>Sunday only</td></tr><tr><td>12</td><td>Monday only</td></tr><tr><td>13</td><td>Tuesday only</td></tr><tr><td>14</td><td>Wednesday only</td></tr><tr><td>15</td><td>Thursday only</td></tr><tr><td>16</td><td>Friday only</td></tr><tr><td>17</td><td>Saturday only</td></tr></table> <p>Weekend string values are seven characters long and each character in the string represents a day of the week, starting with Monday. 1 represents a non-workday and 0 represents a workday. Only the characters 1 and 0 are permitted in the string. For example, 0000110 would result in a weekend that is Friday and Saturday.</p> <p>Holidays is the optional parameter. It is one or more dates that are to be excluded from the working days. Holidays should be a range</p>	Weekend Number	Weekend Days	1 or omitted	Saturday, Sunday	2	Sunday, Monday	3	Monday, Tuesday	4	Tuesday, Wednesday	5	Wednesday, Thursday	6	Thursday, Friday	7	Friday, Saturday	11	Sunday only	12	Monday only	13	Tuesday only	14	Wednesday only	15	Thursday only	16	Friday only	17	Saturday only
Weekend Number	Weekend Days																															
1 or omitted	Saturday, Sunday																															
2	Sunday, Monday																															
3	Monday, Tuesday																															
4	Tuesday, Wednesday																															
5	Wednesday, Thursday																															
6	Thursday, Friday																															
7	Friday, Saturday																															
11	Sunday only																															
12	Monday only																															
13	Tuesday only																															
14	Wednesday only																															
15	Thursday only																															
16	Friday only																															
17	Saturday only																															

		of cells containing dates, or an array of the serial values.												
NOW	Returns the current date and time as a serial number.	NOW()												
SECOND	Returns an integer that is the seconds component of a specified time.	SECOND(time) Time can be a text string representing time, the result of other functions, such as TIMEVALUE, or a decimal number. The decimal number is calculated as a portion of a date value (24 hours is 1.0, so 3 AM is 0.125 in decimal time).												
TIME	Returns a decimal number that represents a specified time.	TIME(hour, minute, second) Hour is a number from 0 (zero) to 32767. Any value greater than 23 will be divided by 24 and its remainder will be taken as an hour value. Minute is a number from 0 (zero) to 32767. Any value greater than 59 will be converted into hours and minutes. Second is a number from 0 (zero) to 32767. Any value greater than 59 will be converted into hours, minutes and seconds.												
TIMEVALUE	Converts a text representation of time into a decimal number that represents time.	TIMEVALUE(time_text) Date information in time_text is ignored. The decimal number is calculated as a portion of a date value (24 hours is 1.0, so 3 AM is 0.125 in decimal time).												
TODAY	Returns a serial number of the current date.	TODAY()												
WEEKDAY	Returns an integer representing the day of the week for a specified date.	WEEKDAY(date_serial_number,[return_type]) Date_serial_number is a serial number representing a date. Use functions such as DATE to supply an argument to the function. Return_type is a number that specifies which integers are to be assigned to each weekday. <table><tr><th>Return_type</th><th>Number returned</th></tr><tr><td>1 or omitted</td><td>Numbers 1 (Sunday) through 7 (Saturday).</td></tr><tr><td>2</td><td>Numbers 1 (Monday) through 7 (Sunday).</td></tr><tr><td>3</td><td>Numbers 0 (Monday) through 6 (Sunday).</td></tr><tr><td>11</td><td>Numbers 1 (Monday) through 7 (Sunday).</td></tr><tr><td>12</td><td>Numbers 1 (Tuesday) through 7 (Monday).</td></tr></table>	Return_type	Number returned	1 or omitted	Numbers 1 (Sunday) through 7 (Saturday).	2	Numbers 1 (Monday) through 7 (Sunday).	3	Numbers 0 (Monday) through 6 (Sunday).	11	Numbers 1 (Monday) through 7 (Sunday).	12	Numbers 1 (Tuesday) through 7 (Monday).
Return_type	Number returned													
1 or omitted	Numbers 1 (Sunday) through 7 (Saturday).													
2	Numbers 1 (Monday) through 7 (Sunday).													
3	Numbers 0 (Monday) through 6 (Sunday).													
11	Numbers 1 (Monday) through 7 (Sunday).													
12	Numbers 1 (Tuesday) through 7 (Monday).													

		<table><tr><td>13</td><td>Numbers 1 (Wednesday) through 7 (Tuesday).</td></tr><tr><td>14</td><td>Numbers 1 (Thursday) through 7 (Wednesday).</td></tr><tr><td>15</td><td>Numbers 1 (Friday) through 7 (Thursday).</td></tr><tr><td>16</td><td>Numbers 1 (Saturday) through 7 (Friday).</td></tr><tr><td>17</td><td>Numbers 1 (Sunday) through 7 (Saturday).</td></tr></table>	13	Numbers 1 (Wednesday) through 7 (Tuesday).	14	Numbers 1 (Thursday) through 7 (Wednesday).	15	Numbers 1 (Friday) through 7 (Thursday).	16	Numbers 1 (Saturday) through 7 (Friday).	17	Numbers 1 (Sunday) through 7 (Saturday).																							
13	Numbers 1 (Wednesday) through 7 (Tuesday).																																		
14	Numbers 1 (Thursday) through 7 (Wednesday).																																		
15	Numbers 1 (Friday) through 7 (Thursday).																																		
16	Numbers 1 (Saturday) through 7 (Friday).																																		
17	Numbers 1 (Sunday) through 7 (Saturday).																																		
WEEKNUM	Returns an integer that is the week number of a specified date.	<p>WEEKNUM(date_serial_number,[return_type])</p> <p>Date_serial_number is a serial number representing a date. Use functions such as DATE to supply an argument to the function.</p> <p>Return_type is a number that specifies a numbering system to use and a weekday that should be treated as the start of the week.</p> <p>There are two systems used for this function. System one defines a week containing January 1 as the first week of the year. System two defines a week containing the first Thursday of the year as the first week of the year (ISO 8601, European week numbering system).</p> <table><tr><th>Return_type</th><th>Week begins on</th><th>System</th></tr><tr><td>1 or omitted</td><td>Sunday</td><td>1</td></tr><tr><td>2</td><td>Monday</td><td>1</td></tr><tr><td>11</td><td>Monday</td><td>1</td></tr><tr><td>12</td><td>Tuesday</td><td>1</td></tr><tr><td>13</td><td>Wednesday</td><td>1</td></tr><tr><td>14</td><td>Thursday</td><td>1</td></tr><tr><td>15</td><td>Friday</td><td>1</td></tr><tr><td>16</td><td>Saturday</td><td>1</td></tr><tr><td>17</td><td>Sunday</td><td>1</td></tr><tr><td>21</td><td>Monday</td><td>2</td></tr></table>	Return_type	Week begins on	System	1 or omitted	Sunday	1	2	Monday	1	11	Monday	1	12	Tuesday	1	13	Wednesday	1	14	Thursday	1	15	Friday	1	16	Saturday	1	17	Sunday	1	21	Monday	2
Return_type	Week begins on	System																																	
1 or omitted	Sunday	1																																	
2	Monday	1																																	
11	Monday	1																																	
12	Tuesday	1																																	
13	Wednesday	1																																	
14	Thursday	1																																	
15	Friday	1																																	
16	Saturday	1																																	
17	Sunday	1																																	
21	Monday	2																																	
WORKDAY	Returns a date that is a specified number of working days (excluding weekends and holidays) before or after a start date.	<p>WORKDAY(start_date, days, holidays)</p> <p>All weekdays are counted; weekends and dates identified as holidays are excluded. To specify holidays to exclude, use an optional third argument that is a range of cells</p>																																	

		containing dates or an array of serial numbers representing dates.												
WORKDAY.INTL	Returns the serial number of the date before or after a specified number of workdays with custom weekend parameters.	WORKDAY.INTL(start_date, days, weekend, holidays) Start_date is a required parameter that specifies the start date. Days is a required parameter that specifies the number of workdays before or after the start_date. Weekend and holidays are optional parameters. For information on these parameters refer to the NETWORKDAYS.INTL description.												
YEAR	Returns an integer that is the year component of a specified date.	YEAR(date_serial_number) The year is returned as an integer in the range 1900-9999.												
YEARFRAC	Returns the year fraction representing the number of whole days between start_date and end_date.	YEARFRAC(start_date, end_date, basis) The basis is an optional parameter that specifies the type of day count. Basis values are listed in the following table. <table><tr><th>Basis</th><th>Day count basis</th></tr><tr><td>0 or omitted</td><td>US (NASD) 30/360</td></tr><tr><td>1</td><td>Actual/actual</td></tr><tr><td>2</td><td>Actual/360</td></tr><tr><td>3</td><td>Actual/365</td></tr><tr><td>4</td><td>European 30/360</td></tr></table>	Basis	Day count basis	0 or omitted	US (NASD) 30/360	1	Actual/actual	2	Actual/360	3	Actual/365	4	European 30/360
Basis	Day count basis													
0 or omitted	US (NASD) 30/360													
1	Actual/actual													
2	Actual/360													
3	Actual/365													
4	European 30/360													

See Also
[How to: Use Functions and Nested Functions in Formulas](#)

Text Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Text Functions](#)

This document describes text functions supported by a non-visual spreadsheet component.

Name	Description	Syntax
BAHTTEXT	Converts a number to text, using the	

Financial Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Financial Functions](#)

This document describes financial functions supported by a non-visual spreadsheet component.

Name	Description	Syntax
ACCRINTM	Returns the accrued interest for a security that pays interest at maturity.	ACCRINTM(issue, settlement, rate, par, [basis])
COUPDAYBS	Returns the number of days from the beginning of the coupon period to the settlement date.	COUPDAYBS(settlement, maturity, frequency, [basis])
COUPNCD	Returns the next coupon date after the settlement date.	COUPNCD(settlement, maturity, frequency, [basis])
COUPNUM	Returns the number of coupons payable between the settlement date and maturity date.	COUPNUM(settlement, maturity, frequency, [basis])
COUPPCD	Returns the previous coupon date before the settlement date.	COUPPCD(settlement, maturity, frequency, [basis])
CUMIPMT	Calculates the cumulative interest paid between two specified periods.	CUMIPMT(rate, nper, pv, start_period, end_period, type)
CUMPRINC	Calculates the cumulative principal paid on a loan, between two specified periods.	CUMPRINC(rate, nper, pv, start_period, end_period, type)
DB	Returns the depreciation of an asset for a specified period by using the fixed-declining balance method.	DB(cost, salvage, life, period, [month])
DDB	Returns the depreciation of an asset for a specified period by using the double-declining balance method or some other method that you specify.	DDB(cost, salvage, life, period, [factor])
DISC	Returns the discount rate for a security.	DISC(settlement, maturity, pr, redemption, [basis])
DOLLARDE	Converts a dollar price, expressed as a fraction, into a dollar price, expressed as a decimal number.	DOLLARDE(fractional_dollar, fraction)
DOLLARFR	Converts a dollar price, expressed as a decimal number, into a dollar price, expressed as a fraction.	DOLLARFR(decimal_dollar, fraction)
FVSCHEDULE	Returns the future value of an initial principal after applying a series of compound interest rates.	FVSCHEDULE(principal, schedule)
EFFECT	Returns the effective annual interest rate.	EFFECT(nominal_rate, npery)
FV	Calculates the future value of an investment with periodic constant payments and a constant interest rate.	FV(rate,nper,pmt,[pv],[type])
INTRATE	Returns the interest rate for a fully invested security.	INTRATE(settlement, maturity, investment, redemption, [basis])

IPMT	Calculates the interest payment for a given period of an investment, with periodic constant payments and a constant interest rate.	IPMT(rate, per, nper, pv, [fv], [type])
IRR	Calculates the internal rate of return for a series of cash flows.	IRR(values, [guess])
ISPMT	Calculates the interest paid during a specific period of an investment.	ISPMT(rate, per, nper, pv)
MIRR	Calculates the internal rate of return for a series of periodic cash flows, considering the cost of the investment and the interest on the reinvestment of cash.	MIRR(values, finance_rate, reinvest_rate)
NOMINAL	Returns the annual nominal interest rate.	NOMINAL(effect_rate, npery)
NPER	Returns the number of periods for an investment with periodic constant payments and a constant interest rate.	NPER(rate, pmt, pv, [fv], [type])
NPV	Calculates the net present value of an investment, based on a supplied discount rate, and a series of future payments and income.	NPV(rate, value1, [value2], ...)
PDURATION	Returns the number of periods required by an investment to reach a specified value.	PDURATION(rate, pv, fv)
PMT	Calculates the payments required to reduce a loan, from a supplied present value to a specified future value.	PMT(rate, nper, pv, [fv], [type])
PPMT	Calculates the payment on the principal for a given investment, with periodic constant payments and a constant interest rate.	PPMT(rate, per, nper, pv, [fv], [type])
PRICEDISC	Returns the price per \$100 face value of a discounted security.	PRICEDISC(settlement, maturity, discount, redemption, [basis])
PV	Calculates the present value of an investment (i.e., the total amount that a series of future payments is worth now).	PV(rate, nper, pmt, [fv], [type])
RATE	Calculates the interest rate required to pay off a specified amount of a loan, or reach a target amount on an investment over a given period.	RATE(nper, pmt, pv, [fv], [type], [guess])
RECEIVED	Returns the amount received at maturity for a fully invested security.	RECEIVED(settlement, maturity, investment, discount, [basis])
RRI	Returns an equivalent interest rate for the growth of an investment.	RRI(nper, pv, fv)
SLN	Returns the straight-line depreciation of an asset for one period.	SLN(cost, salvage, life)
SYD	Returns the sum-of-years' digits depreciation of an asset for a specified period.	SYD(cost, salvage, life, per)

TBILLEQ	Returns the bond-equivalent yield for a Treasury bill.	TBILLEQ(settlement, maturity, discount)
TBILLPRICE	Returns the price per \$100 face value for a Treasury bill.	TBILLPRICE(settlement, maturity, discount)
TBILLYIELD	Returns the yield for a Treasury bill.	TBILLYIELD(settlement, maturity, pr)
VDB	Returns the depreciation of an asset for a specified or partial period by using a declining balance method	VDB(cost, salvage, life, start_period, end_period, [factor], [no_switch]).
YIELDDISC	YIELDDISC(settlement, maturity, pr, redemption, [basis])	Returns the annual yield for a discounted security; for example, a Treasury bill.
XNPV	Returns the net present value for a schedule of cash flows that is not necessarily periodic.	XNPV(rate, values, dates)
XIRR	Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.	XIRR(values, dates, [guess])

Logical Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Logical Functions](#)

This document describes the logical functions supported by a non-visual spreadsheet component.

Name	Description	Syntax
AND	Returns TRUE if all of its arguments are TRUE; returns FALSE if one or more arguments evaluate to FALSE.	AND(logical1, [logical2], ...)
FALSE	Returns the logical value FALSE.	FALSE()
IF	Checks a condition and returns one result if the condition is TRUE, and another result if the condition is FALSE.	IF(logical_test, [value_if_true], [value_if_false])
IFERROR	Checks whether or not a value (or expression) returns an error, and if so, the function returns a specified value; otherwise, the function returns the initial value.	IFERROR(value, value_if_error)
IFNA	Checks whether a value (or expression) resolves to #N/A, and if so the function returns a specified value; otherwise the function returns the initial value.	IFNA(value, value_if_na)
NOT	Reverses the logical value of its argument.	NOT(logical)
OR	Returns TRUE if any argument is TRUE; returns FALSE if all arguments are FALSE.	OR(logical1, [logical2], ...)
TRUE	Returns the logical value TRUE.	TRUE()
XOR	Returns a logical exclusive OR of all arguments.	XOR(logical1, [logical2], ...)

Lookup and Reference Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Lookup and Reference Functions](#)

This document describes the lookup and reference functions supported by a non-visual spreadsheet component.

Name	Description	Syntax
ADDRESS	Returns a text reference for a specified row and column number.	ADDRESS(row_num, column_num, [abs_num], [a1], [sheet_text])
AREAS	Returns the number of areas in a range.	AREAS(reference)
CHOOSE	Selects a value from a list based on its index number.	CHOOSE(index_num, value1, [value2], ...)
COLUMN	Returns the first column number within the cell reference or the number of the current column if no reference is supplied.	COLUMN([reference])
COLUMNS	Returns the number of columns in an array or reference.	COLUMNS(array)
FORMULATEXT	Returns what is displayed in the formula bar if you select the referenced cell.	FORMULATEXT(reference)
HLOOKUP	Looks up a value in the first table row, and returns a value in the same column from another row.	HLOOKUP(lookup_value, table_array, row_index_num, [range_lookup])
HYPERLINK	Creates a hyperlink.	HYPERLINK(link_location,friendly_name)
INDEX	Returns the value of an element in a table or an array, selected by the row and column number indexes.	INDEX(array, row_num, [column_num])
INDIRECT	Returns the reference specified by a text string.	INDIRECT(ref_text, [a1])
LOOKUP	Returns a value from a cell in a position found by lookup in a search table.	<p>Vector form: LOOKUP(lookup_value, lookup_vector, result_vector)</p> <p>'lookup_value' is the value to look up in the 'lookup_vector' single column (single row) range, 'lookup_vector' is a list of data (single column or row range) used to search for the lookup_value; 'result_vector' is a range of the same size as 'lookup_vector'. The function returns the value in 'result_vector' at the position where the match is found in 'lookup_vector'.</p> <p>Array form: LOOKUP(lookup_value, array)</p> <p>'lookup_value' is the value that you wish to look up in the specified array and 'array' is a two-dimensional array of data. The first column (or row) of an array will be used to search for the 'lookup_value', and the value in the corresponding last column (or row) will be returned.</p>

MATCH	Searches for a specified item in a range of cells, and returns the relative position of that item in the range.	MATCH(lookup_value, lookup_array, [match_type])
OFFSET	Returns a reference to a range that is located a specified number of rows and columns away from a cell or range of cells.	OFFSET(reference, rows, cols, [height], [width])
ROW	Returns the first row number within the cell reference or the number of the current row if no reference is supplied.	ROW([reference])
ROWS	Returns the number of rows in an array or reference.	ROWS(array)
TRANSPOSE	Transforms a horizontal range of cells into a vertical range, and vice versa.	TRANSPOSE(array)
VLOOKUP	Looks up a value in the first column of a table, and returns a value in the same row from another column.	VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])

Engineering Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Engineering Functions](#)

This document describes the engineering functions supported by a non-visual spreadsheet component.

Name	Description	Syntax
BESSELI	Returns the modified Bessel function $I_n(x)$.	BESSELI(X, N)
BESSELJ	Calculates the Bessel function $J_n(x)$.	BESSELJ(X, N) X is the value at which to evaluate the function. N is the order of the Bessel function. If n is not an integer, it is truncated.
BESSELK	Returns the modified Bessel function $K_n(x)$.	BESSELK(X, N)
BESSELY	Returns the Bessel function $Y_n(x)$.	BESSELY(X, N)
BIN2DEC	Converts a binary number to decimal.	BIN2DEC(number)
BIN2HEX	Converts a binary number to hexadecimal.	BIN2HEX(number, [places]) Number is the binary number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.
BIN2OCT	Converts a binary number to octal.	BIN2OCT(number, [places]) Number is the binary number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.
BITAND	Returns a 'Bitwise And' of two numbers.	BITAND(number1, number2)
BITLSHIFT	Returns a value number shifted left by shift_amount bits.	BITLSHIFT(number, shift_amount)
BITOR	Returns a bitwise OR of 2 numbers.	BITOR(number1, number2)
BITRSHIFT	Returns a value number shifted right by shift_amount bits.	BITRSHIFT(number, shift_amount)
BITXOR	Returns a bitwise 'Exclusive Or' of two numbers.	BITXOR(number1, number2)
COMPLEX	Converts real and imaginary coefficients into a complex number of the form $x + yi$ or $x + yj$.	COMPLEX(real_num,i_num,suffix) Real_num is the real coefficient of the complex number. I_num is the imaginary coefficient of the complex number. Suffix is the suffix for the imaginary component of the complex number. If omitted, suffix is assumed to be "i".
CONVERT	Converts a number from one measurement system to another.	CONVERT(number, from_unit, to_unit) Number is the number to be converted. From_unit is a text string, denoting the current unit. To_unit is a text string, denoting the unit to which the number will be converted.

DEC2BIN	Converts a decimal number to binary.	DEC2BIN(number, [places]) Number is the decimal number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.
DEC2HEX	Converts a decimal number to hexadecimal.	DEC2HEX(number, [places]) Number is the decimal number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.
DEC2OCT	Converts a decimal number to octal.	DEC2OCT(number, [places]) Number is the decimal number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.
DELTA	Tests whether two values are equal.	DELTA(number1, number2)
ERF	Returns the error function integrated between lower_limit and upper_limit.	ERF(lower_limit,[upper_limit])
ERFC	Returns the complementary ERF function integrated between x and infinity.	ERFC(x)
ERFC.PRECISE	Returns the complementary ERF function integrated between x and infinity.	ERFC.PRECISE(x)
ERF.PRECISE	Returns the error function integrated between 0 and a specified limit.	ERF.PRECISE(x)
GESTEP	Returns 1 if number = step; returns 0 (zero) otherwise. Use this function to filter a set of values.	GESTEP(number,step)
HEX2BIN	Converts a hexadecimal number to binary.	HEX2BIN(number, [places]) Number is the decimal number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.
HEX2DEC	Converts a hexadecimal number to decimal	HEX2DEC(number)
HEX2OCT	Converts a hexadecimal number to octal.	HEX2OCT(number, [places]) Number is the decimal number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.
IMABS	Returns the absolute value (the modulus) of a complex number.	IMABS(inumber)
IMAGINARY	Returns the imaginary coefficient of a complex number in x + yi or x + yj text format.	IMAGINARY(inumber)
IMARGUMENT	Returns the argument (an angle expressed in radians) of a complex number.	IMARGUMENT(inumber)

IMCONJUGATE	Returns the complex conjugate of a complex number in $x + yi$ or $x + yj$ text format.	IMCONJUGATE(inumber)
IMCOS	Returns the cosine of a complex number	IMCOS(inumber)
IMCOSH	Returns the hyperbolic cosine of a complex number.	IMCOSH(inumber)
IMCOT	Returns the cotangent of a complex number.	IMCOT(inumber)
IMCSC	Returns the cosecant of a complex number.	IMCSC(inumber)
IMCSCH	Returns the hyperbolic cosecant of a complex number.	IMCSCH(inumber)
IMDIV	Returns the quotient of two complex numbers in $x + yi$ or $x + yj$ text format.	IMDIV(inumber1,inumber2)
IMEXP	Returns the exponential of a complex number in $x + yi$ or $x + yj$ text format.	IMEXP(inumber)
IMLN	Returns the natural logarithm of a complex number in $x + yi$ or $x + yj$ text format.	IMLN(inumber)
IMLOG10	Returns the common logarithm (base 10) of a complex number in $x + yi$ or $x + yj$ text format.	IMLOG10(inumber)
IMLOG2	Returns the base-2 logarithm of a complex number in $x + yi$ or $x + yj$ text format.	IMLOG2(inumber)
IMPOWER	Returns a complex number raised in the specified power.	IMPOWER(inumber, number)
IMPRODUCT	Returns the product of complex numbers in $x + yi$ or $x + yj$ text format.	IMPRODUCT(inumber1,inumber2,...)
IMREAL	Returns the real coefficient of a complex number in $x + yi$ or $x + yj$ text format.	IMREAL(inumber)
IMSEC	Returns the secant of a complex number.	IMSEC(inumber)
IMSECH	Returns the hyperbolic secant of a complex number.	IMSECH(inumber)
IMSIN	Returns the sine of a complex number in $x + yi$ or $x + yj$ text format.	IMSIN(inumber)
IMSINH	Returns the hyperbolic sine of a complex number.	IMSINH(inumber)
IMSQRT	Returns the common logarithm (base 10) of a complex number in $x + yi$ or $x + yj$ text format.	IMSQRT(inumber)
IMSUB	Returns the difference of two complex numbers in $x + yi$ or $x + yj$ text format.	IMSUB(inumber1,inumber2)

IMSUM	Returns the sum of complex numbers.	IMSUM(inumber1, inumber2, ...)
IMTAN	Returns the tangent of a complex number.	IMTAN(number)
OCT2BIN	Converts an octal number to binary.	OCT2BIN(number, [places]) Number is the octal number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.
OCT2DEC	Converts an octal number to decimal.	OCT2DEC(number)
OCT2HEX	Converts an octal number to hexadecimal.	OCT2HEX(number, [places]) Number is the octal number to be converted. Places is the number of characters to use. If places is omitted, the minimum number of characters is used.

Information Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Information Functions](#)

This document describes information functions supported by a non-visual spreadsheet component.

Name	Description	Syntax
CELL	Returns information about the formatting, location, or content of a cell.	CELL(info_type, [reference])
ERROR.TYPE	Returns a number corresponding to one of the error values.	ERROR.TYPE(value)
INFO	Returns information about the current operating environment.	INFO(type_text)
ISBLANK	Tests if the value is a reference to an empty cell.	ISBLANK(value)
ISERR	Tests if the value or expression returns an error.	ISERR(value)
ISERROR	Tests if the value or expression returns an error, except for the #N/A error.	ISERROR(value)
ISEVEN	Tests if the number is even.	ISEVEN(value)
ISFORMULA	Tests if the referenced cell contains formula.	ISFORMULA(reference)
ISLOGICAL	Tests if the value is a logical value.	ISLOGICAL(value)
ISNA	Tests if the value or expression returns the #N/A error.	ISNA(value)
ISNONTEXT	Tests if the value or expression is text.	ISNONTEXT(value)
ISNUMBER	Tests if the value or expression is a number.	ISNUMBER(value)
ISODD	Tests if the number is odd.	ISODD(value)
ISREF	Tests if the value is a reference.	ISREF(value)
ISTEXT	Tests if the value or expression is text.	ISTEXT(value)
N	Returns a value converted to a number.	N(value)
NA	Returns the error value #N/A.	NA()
SHEET	Returns the sheet number of the referenced sheet.	SHEET(value) The value parameter is optional. It is the name of a sheet or a reference for which to obtain the sheet number. If the value is omitted, the number of the sheet that contains the function is returned.
SHEETS	Returns the number of sheets in a reference.	SHEETS(reference) The reference parameter is optional. If the parameter is omitted, the number of sheets in the workbook that contains the function is returned.

TYPE	Returns the type of value.	TYPE(value)
------	----------------------------	-------------

Compatibility Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Compatibility Functions](#)

This document describes the compatibility functions supported by a non-visual spreadsheet component.

Compatibility functions are functions that were replaced with new functions in Microsoft Excel versions 2010 and higher. These functions, however, are still available for use in documents created with earlier versions of Excel. Our spreadsheet correctly interprets and evaluates them.

Consider using the new, more precise functions instead of the functions listed in this document.

Name	Description	Syntax
BETADIST	Returns the beta cumulative distribution function.	BETADIST(x,alpha,beta,[A],[B])
BETAINV	Returns the inverse of the cumulative beta probability density function for a specified beta distribution.	BETAINV(probability,alpha,beta,A,B)
BINOMDIST	Returns the individual term binomial distribution probability.	BINOMDIST(number_s,trials,probability_s,cumulative)
CHIDIST	Returns the one-tailed probability of the chi-squared distribution.	CHIDIST(x,degrees_freedom)
CHIINV	Returns the inverse of the one-tailed probability of the chi-squared distribution.	CHIINV(probability,degrees_freedom)
CHITEST	Returns the test for independence as the value from the chi-squared distribution for the statistic and the appropriate degrees of freedom	CHITEST(actual_range,expected_range)
CONFIDENCE	Returns the confidence interval for a population mean	CONFIDENCE(alpha,standard_dev,size)
COVAR	Returns covariance, the average of the products of deviations for each data point pair in two data sets.	COVAR(array1,array2)
CRITBINOM	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value.	CRITBINOM(trials,probability_s,alpha)
EXPONDIST	Returns the exponential distribution.	EXPONDIST(x,lambda,cumulative)
FDIST	Returns the F probability distribution.	FDIST(x,degrees_freedom1,degrees_freedom2)
FINV	Returns the inverse of the F probability distribution.	FINV(probability,degrees_freedom1,degrees_freedom2)
FTEST	Returns the two-tailed probability that the variances in array1 and array2 are not significantly different.	FTEST(array1,array2)
GAMMADIST	Returns the gamma distribution.	GAMMADIST(x,alpha,beta,cumulative)
GAMMAINV	Returns the inverse of the gamma cumulative distribution.	GAMMAINV(probability,alpha,beta)
HYPGEOMDIST	Returns the hypergeometric distribution.	HYPGEOMDIST(sample_s,number_sample,population_s,number_pop)

LOGINV	Returns the inverse of the lognormal cumulative distribution function	LOGINV(probability,mean,standard_dev)
LOGNORMDIST	Returns the cumulative lognormal distribution	LOGNORMDIST(x,mean,standard_dev)
MODE	Returns the most frequently occurring, or repetitive, value.	MODE(number1,[number2],...)
NEGBINOMDIST	Returns the negative binomial distribution.	NEGBINOMDIST(number_f,number_s,probability_s)
NORMDIST	Returns the normal distribution for the specified mean and standard deviation.	NORMDIST(x,mean,standard_dev,cumulative)
NORMINV	Returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.	NORMINV(probability,mean,standard_dev)
NORMSDIST	Returns the standard normal cumulative distribution function.	NORMSDIST(z)
NORMSINV	Returns the inverse of the standard normal cumulative distribution.	NORMSINV(probability)
PERCENTILE	Returns the k-th percentile of values in a range.	PERCENTILE(array,k)
PERCENTRANK	Returns the percentage rank of a value in a data set.	PERCENTRANK(array,x,[significance])
POISSON	Returns the Poisson distribution.	POISSON(x,mean,cumulative)
QUARTILE	Returns the quartile of a data set.	QUARTILE(array,quart)
RANK	Returns the rank of a number in a list of numbers.	RANK(number,ref,[order])
STDEV	Calculates standard deviation based on a sample.	STDEV(number1,[number2],...)
STDEVP	Calculates standard deviation based on the entire population.	STDEVP(number1,[number2],...)
TINV	Returns the t-value of the Student's t-distribution as a function of the probability and the degrees of freedom.	TINV(probability,degrees_freedom)
TDIST	Returns the Percentage Points (probability) for the Student t-distribution where a numeric value (x) is a calculated value of t for which the Percentage Points are to be computed.	TDIST(x,degrees_freedom,tails)
TTEST	Returns the probability associated with a Student's t-Test.	TTEST(array1,array2,tails,type)
VAR	Calculates variance based on a sample.	VAR(number1,[number2],...)
VARP	Calculates variance based on the entire population.	VARP(number1,[number2],...)
WEIBULL	Returns the Weibull distribution.	WEIBULL(x,alpha,beta,cumulative)
ZTEST	Returns the one-tailed probability-value of a z-test.	ZTEST(array,x,[sigma])

Database Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Database Functions](#)

This document describes the database functions supported by a non-visual spreadsheet component.

The database functions operates on a range of cells that makes up the list or database. A database in this context is a list of related data in which rows are records and columns are fields. The first row of the list contains labels for each column.

The database (or list) is identified as the *datalist* parameter in function syntax.

Name	Description	Syntax
DAVERAGE	Returns the average of selected database entries.	DAVERAGE(datalist, field, criteria)
DCOUNT	Counts the cells that contain numbers in a database.	DCOUNT(datalist, field, criteria)
DCOUNTA	Counts nonblank cells in a database.	DCOUNTA(datalist, field, criteria)
DGET	Extracts from a database a single record that matches the specified criteria.	DGET(datalist, field, criteria)
DMAX	Returns the maximum value from selected database entries.	DMAX(datalist, field, criteria)
DMIN	Returns the minimum value from selected database entries.	DMIN(datalist, field, criteria)
DPRODUCT	Multiplies the values in a particular field of records that match the criteria in a database.	DPRODUCT(datalist, field, criteria)
DSTDEV	Estimates the standard deviation based on a sample of selected database entries.	DSTDEV(datalist, field, criteria)
DSTDEVP	Calculates the standard deviation based on the entire population of selected database entries.	DSTDEVP(datalist, field, criteria)
DSUM	Adds the numbers in the field column of records in the database that match the criteria.	DSUM(datalist, field, criteria)
DVAR	Estimates variance based on a sample from selected database entries.	DVAR(datalist, field, criteria)
DVARP	Calculates variance based on the entire population of selected database entries.	DVARP(datalist, field, criteria)

Web Functions

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Web Functions](#)

This document describes the web functions supported by a non-visual spreadsheet component.

Name	Description	Syntax
ENCODEURL	Returns a URL-encoded string.	ENCODEURL(text)

User-Defined Functions (UDF)

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [User-Defined Functions \(UDF\)](#)

Overview

The Spreadsheet API provides the capability to create your own custom functions. Custom functions are available for spreadsheet calculations, and can be used in formulas in the same manner as built-in functions. The main difference is that custom functions are not saved in a workbook file. They should be added in code to the [Workbook.CustomFunctions](#) or the [Workbook.GlobalCustomFunctions](#) collection.

Note

Add a custom function to the CustomFunctionCollection collection **before** loading a document.

If a worksheet contains a custom function that is not recognized by the workbook (or MS Excel), the "#NAME!" error is displayed after the cell containing the function is recalculated. To replace custom function definitions with the corresponding calculated values when saving a workbook, set the WorkbookExportOptions.CustomFunctionExportMode option to CustomFunctionExportMode.CalculatedValue.

Limitations

A custom function called in a worksheet cell should not change the properties and characteristics of the worksheet. The IFunction.Evaluate method of the function has access to the EvaluationContext object, which provides information about the current worksheet and workbook. However, do not call methods or set properties that might perform the following actions:

- Insert, delete, or format cells;
- Move, rename, delete, or add sheets to a workbook;
- Add names to a workbook;
- Change the values of other cells.

Implementation

A custom function is an object that exposes the ICustomFunction interface. To create a custom function, inherit from this interface and implement the required properties and methods. You should specify the IFunction.Name, and input IFunction.Parameters and IFunction.ReturnType. All calculations in a custom function are performed in the IFunction.Evaluate method. The IFunction.Volatile property specifies whether a cell that contains a custom function should be reevaluated every time a spreadsheet is recalculated.

By specifying the number and type of input parameters, you enable the workbook to validate the formula entered. If the formula is missing required parameters, an error message is displayed.

Custom functions are contained in a CustomFunctionCollection collection, which is available via the IWorkbook.CustomFunctions property. To use a custom function in a worksheet, add an instance of your function to the collection.

Real Time Data (RTD) function

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Functions](#) > [Real Time Data \(RTD\) function](#)

The **Real-Time Data (RTD)** worksheet function enables you to retrieve data in real time from a COM Automation server. For more information on creating a COM Automation server that can be used as a Real-Time Data server for a spreadsheet, refer to the [How to create a RealTimeData server for Excel](#) article in MSDN.

By default, the function result is updated by timer at the interval specified by the RealTimeDataOptions.ThrottleInterval property. To update the result manually, set the RealTimeDataOptions.RefreshMode property to the RealTimeDataRefreshMode.Manual value and call the RealTimeData.RefreshData method when required.

Syntax

RTD(RealTimeServerProgID, ServerName, Topic1, [Topic2], ...)

The following table describes the arguments in the above syntax.

Argument	Description
RealTimeServerProgID	A string that represents the Program ID of the RTD server installed on the local system.
ServerName	A string that represents the name of the server on which the RTD server is run. If the RTD server is run locally, the ServerName is an empty string ("") or can be omitted.
Topic1, [Topic2], ...	Strings that determine the data being retrieved. At least one topic is required.

Operators

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Operators](#)

There are four different types of binary operators: arithmetic, boolean, text concatenation, and reference. Unary operators are represented by unary minus, unary plus and percent operators.

Arithmetic Operators

Before performing a specific action, each arithmetic operator evaluates the operands or converts the operands to numeric values.

If an operand is a string that is not surrounded by quotes, the operator tries to do the following:

- recognize the operand as the defined name, the cell or the array reference, and get the value
- use the **System.Double.TryParse** method to convert a string to a numeric value
- recognize the operand as a percentage value and then convert it to numeric
- recognize the operand as a boolean value and return 0 for *FALSE* and 1 for *TRUE*

If none of these methods succeed, the **Name error** occurs, which displays the "#NAME?" string.

If an operand is a string surrounded by quotes, no assumptions are made about the defined names or references. The operator tries to use methods listed previously to convert the operand to numeric. Also, an attempt is made to recognize the operand as a date or time value, and get the numeric representation of time. When the operand is a quoted string and all attempts fail, the **Invalid Value in Function error** occurs, which displays the "#VALUE!" string. This error also appears when your formula operates with blank cells that are not actually empty (e.g., when an empty string "" is assigned to any cell in a formula). To avoid this error and specify an empty value for a cell, assign the Range.Value property to **null** or CellValue.Empty.

Arithmetic operators are listed in the following table.

Arithmetic Operator	Action and Specifics
+ (plus sign) - binary	Values are added together.
+ (plus sign) - unary	Evaluates the operand if it is a reference, otherwise converts it to numeric.
- (minus sign) - binary	Subtracts the value of the right operand from the value of the left operand.
- (minus sign) - unary	Negates the value of the operand.
* (asterisk)	Multiplies the value of the left operand by the value of the right operand.
/ (forward slash)	Divides the value of the left operand by the value of the right operand. If the right value evaluates to zero, a Division by Zero error occurs and the "#DIV/0!" string is displayed.
^ (caret)	The exponentiation operation. Returns the value of the first operand raised to the power of the second operand value. The operator is equivalent to the POWER() function.

Boolean Operators

Boolean operators are used to compare the values of its operands.

If an operand is a string that is not surrounded by quotes, the operator tries to do the following:

- recognize the operand as the defined name, the cell or the array reference, and get the value
- use the **System.Double.TryParse** method to convert a string to a numeric value
- recognize the operand as a percentage value and then convert it to numeric
- recognize the operand as a boolean value and return 0 for *FALSE* and 1 for *TRUE*

If none of the methods succeed, the **Name error** occurs, which displays the "#NAME?" string. If an operand is a string surrounded by quotes, it is considered a string and a comparison is performed. Strings are compared based on a language-specific sort order. The result of this comparison is a Boolean value (*TRUE* or *FALSE*).

Boolean operators are listed in the following table.

Boolean Operator	Action and Specifics
= (equal sign)	The values are equal.
> (greater than sign)	The left value is greater than the right value.
< (less than sign)	The left value is less than the right value.
>= (greater than sign and equal sign)	The left value is greater than or equal to the right value.
<= (less than sign and equal sign)	The left value is less than or equal to the right value.
<> (less than sign and greater than sign)	The values are not equal.

Text Concatenation Operator

The **& (ampersand)** operator is used to concatenate two operands to produce a single string of text. Before performing concatenation, each operand is evaluated to produce a text value. If the operand is recognized as the defined name, the cell, or array reference, its value is obtained and converted to a string.

For example, the result of the formula **=(5=5)&(5=9)** will be the text string **TRUEFALSE**.

Reference Operators

Reference operators combine cell ranges for subsequent calculations.

Reference operators are listed in the following table.

Reference Operator	Action and Specifics
: (colon)	The Cell Range operator. Produces a reference to all cells located between the cells specified as operands. The cells that are operands are included in the resulting reference.
, (comma)	The Union operator. Combines references to produce a single reference.
(space)	The Intersection operator. Produces a reference to cells that are included in both references.

Array Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Array Formulas](#)

What is an Array Formula

An **Array formula** is a formula that performs actions on two or more sets of values, which are called array arguments. Each array argument must have the same number of rows and columns. The result of an array formula can be either a single value or multiple values.

Array arguments can be cell ranges or *array constants*.

An array formula that returns a single value typically processes a series of data, and aggregates it using the SUM, COUNT or AVERAGE functions. The resulting value is written into a cell to which the formula belongs. Usually, an array formula returns a series of data. Returned values are distributed among the cells of the array formula range. If the number of values exceeds the number of cells, excessive values are not shown. If the number of cells is greater than the number of values, excessive cells are not left blank - they are filled with the same values repeatedly.

An individual cell or group of cells that is part of the array formula range is read-only. Any attempt to change the cell value results in an exception. An array formula range only allows you to change the formula for the entire range. However, you can change cell formatting (color, font, etc.) for each cell individually.

How to Create an Array Formula

This example demonstrates how to create an array formula.

C#

```
// Create an array formula that multiplies values contained in the cell range A2 through A11
// by the corresponding cells in the range B2 through B11,
// and displays the results in cells C2 through C11.
worksheet.Range["C2:C11"].ArrayFormula = "=A2:A11*B2:B11";
// Create an array formula that multiplies values contained in the cell range C2 through C11 by 2
// and displays the results in cells D2 through D11.
worksheet.Range["D2:D11"].ArrayFormula = "=C2:C11*2";
// Create an array formula that multiplies values contained in the cell range B2 through D11,
// adds the results, and displays the total sum in cell D12.
worksheet.Cells["D12"].ArrayFormula = "=SUM(B2:B11*C2:C11*D2:D11)";
// Re-dimension an array formula range:
// delete the array formula and create a new range with the same formula.
if (worksheet.Cells["C13"].HasArrayFormula) {
    string af = worksheet.Cells["C13"].ArrayFormula;
    worksheet.Cells["C13"].GetArrayFormulaRange().ArrayFormula = string.Empty;
    worksheet.Range["C2:C11"].ArrayFormula = af;
}
```

Visual Basic

```
' Create an array formula that multiplies values contained in the cell range A2 through A11
' by the corresponding cells in the range B2 through B11,
' and displays the results in cells C2 through C11.
worksheet.Range("C2:C11").ArrayFormula = "=A2:A11*B2:B11"
' Create an array formula that multiplies values contained in the cell range C2 through C11 by 2
' and displays the results in cells D2 through D11.
worksheet.Range("D2:D11").ArrayFormula = "=C2:C11*2"
' Create an array formula that multiplies values contained in the cell range B2 through D11,
' adds the results, and displays the total sum in cell D12.
worksheet.Cells("D12").ArrayFormula = "=SUM(B2:B11*C2:C11*D2:D11)"
' Re-dimension an array formula range:
' delete the array formula and create a new range with the same formula.
If worksheet.Cells("C13").HasArrayFormula Then
    Dim af As String = worksheet.Cells("C13").ArrayFormula
    worksheet.Cells("C13").GetArrayFormulaRange().ArrayFormula = String.Empty
    worksheet.Range("C2:C11").ArrayFormula = af
End If
```

The Range object provides the Range.ArrayFormula property to specify the array formula for a range of cells. The range of cells containing the same array formula is treated as a single entity. Thus, you can only modify data for the entire range.

To determine the array formula range, use the `Cell.GetArrayFormulaRange` method. If a cell is not a part of the range specified by the array formula, this method returns **null**.

All array formulas in a worksheet are accessible via the `Worksheet.ArrayFormulas` property that returns the `ArrayFormulaCollection` object. This object is a collection of `ArrayFormula` type items. Therefore, you can easily determine the `ArrayFormula.Range` of cells that contain the array formula, as well as the text of the `ArrayFormula.Formula` itself.

How to Modify an Array Formula

Array formulas can be modified for the entire formula range only. To change the formula, get its range via the `Cell.GetArrayFormulaRange` method or the `ArrayFormula.Range` property. You can also access the required range using the `Worksheet.Range` property. The `Range.ArrayFormula` property will allow you to get or set the array formula for the created range.

For each cell that belongs to the array formula range, the `Range.ArrayFormula` property returns a formula string, and throws a **System.InvalidOperationException** when trying to set its value.

To check whether or not the cell is the top left cell in the range, use the `Cell.IsTopLeftCellInArrayFormulaRange` property.

Cells have the `Range.Formula` property, which gets or sets ordinary formulas. If a cell belongs to the array formula range, the `Range.Formula` property returns the same string as the `Range.ArrayFormula` property. An attempt to set the **Formula** property for any cell(s) in the array formula range results in an exception indicating that a portion of the array cannot be changed.

How to Delete an Array Formula

To delete an array formula, assign an empty string to the `Range.ArrayFormula` property of the entire array formula range.

See Also

`Cell.IsTopLeftCellInArrayFormulaRange`

`Worksheet.ArrayFormulas`

`Range.ArrayFormula`

[How to: Create Array Formulas Functions](#)

Formula Engine

[Office File API](#) > [Spreadsheet Document API](#) > [Spreadsheet Formulas](#) > [Formula Engine](#)

Overview

The FormulaEngine is an object that provides the capability to calculate and parse worksheet formulas. It includes a built-in formula parser, as well as the flexibility to evaluate formulas in any range of any worksheet using the FormulaEngine.Evaluate method. You can parse a formula using the FormulaEngine.Parse method, analyze the resulting expression tree, make the required modifications and rebuild the formula string from the modified expression tree.

Note

You can get a parsed expression of a formula contained in a cell by using the Cell.ParsedExpression property. To obtain a parsed expression of a formula specified by defined name, use the DefinedName.ParsedExpression property.

Context

The formula context determines the environment that affects the function name recognition, formula calculation and reference resolution. The context is defined by the cell or range to which the formula belongs, the worksheet, the culture settings and the cell reference style.

The ExpressionContext object serves as a container to hold context settings.

You can pass the context to the FormulaEngine.Evaluate or FormulaEngine.Parse method. If the context is not specified explicitly, the currently selected range, active cell and active worksheet are used.

Expression

The ParsedExpression object is the result of parsing an expression by using the FormulaEngine.Parse method. An expression is parsed into an expression tree, which is made available by using the ParsedExpression.Expression property.

The nodes in the expression tree implement the IExpression interface. The technique to traverse the tree is based on the **Visitor pattern**. The Visitor pattern is briefly explained below.

To traverse the tree and visit all nodes, we need a starting (entry) point and a visitor. The entry point can be any tree node, but the root node would be an obvious choice. The root node is accessible from the ParsedExpression.Expression property. To start traversing, call the IExpression.Visit method of the node with the visitor as the method parameter. The node will subsequently call the **Visit** method of the visitor with the node itself as the parameter. Thus, the **Visit** method overload is called, which has a parameter type that is equal to the type of the node. Then, the visitor's **Visit** method is called for sibling nodes if they exist.

Take, for example, the formula SUM(A4:A6)+10. The root node is the AdditionExpression (corresponding to the $+$ operand in the formula) and the Visit(AdditionExpression expression) method of the visitor is called. This method calls the VisitBinary(BinaryOperatorExpression expression) method, which in turn, calls the Visit(FunctionExpression expression) method for the left operand (the **SUM** function). This method calls the Visit(CellReferenceExpression expression) for the function argument, which is the **A4:A6** cell range reference. The node containing the cell reference has no siblings, so all nodes of the left branch of the root addition expression are visited, and the **Visit** method of the right operand is called. The right operand is the constant value **10**, which has no child branches. The Visit(DevExpress.Spreadsheet.Formulas.ConstantExpression expression) method overload is called for that node. Expression tree iteration is now complete.

To create a string formula from the expression tree, use the ParsedExpression.ToString method.

It is often useful to be able to refer to the cells whose values are used in the formula. The ParsedExpression.GetRanges method returns a collection of ranges referenced by the formula.

Visitor

A *visitor* is required to visit all nodes of the expression tree. Implement a descendant of the ExpressionVisitor class and override the ExpressionVisitor.Visit method overloads, which obtain nodes of the required types. After obtaining the node using the visitor, you can modify node properties (e.g., change cell reference) or navigate to child nodes using UnaryOperatorExpression.InnerExpression, BinaryOperatorExpression.LeftExpression or BinaryOperatorExpression.RightExpression, depending on the type of the obtained node.

Defined Names

[Office File API](#) > [Spreadsheet Document API](#) > [Defined Names](#)

This document introduces the **Defined Name** concept and describes how to manage defined names.

- [Defined Name Overview](#)
- [Use Defined Name to Access a Range](#)
- [Defined Name Scope](#)
- [Syntax Rules for Names](#)
- [Create Defined Names](#)
- [Access and Change Defined Names](#)
- [Delete Defined Names](#)

Defined Name Overview

To make it easier to understand the information contained in a worksheet and refer to [individual cells, ranges of cells, formulas](#) and constant values, you can use *defined names*. A defined name is an object that implements the DefinedName interface and contains the following information.

Property	Description
Name DefinedName.Name	Indicates an individual cell, range of cells, formula or constant. Usually, a name explains the purpose of an object to which this name refers, making it easier to find and use this object. When specifying a name, you must take into account special syntax rules .
Refers To DefinedName.RefersTo	A string specifying a reference to a cell or cell range, formula or constant associated with the defined name. For example: "=Sheet1!\$D\$20" - refers to the D20 cell located on the Sheet1 worksheet; "=Sheet1!\$A\$1:\$C\$10" - refers to the A1:C10 range of cells located on the Sheet1 worksheet; "=SUM(Sheet1!\$B\$1:\$B\$10)" - refers to the formula that calculates the sum of values contained in the B1:B10 range of cells located on the Sheet1 worksheet; "=10.5" - refers to a constant value. By default, defined names use absolute cell references , including worksheet names.
Comment DefinedName.Comment	An explanation or additional information accompanying the defined name. <div><div>Note</div><div>The comment length cannot exceed 255 characters.</div></div>

Use Defined Name to Access a Range

To access a worksheet range with a defined name, use the Worksheet.Item or IWorkbook.Range property, as illustrated in the following code snippet:

C#

DevExpress.Spreadsheet.Range myRange = workbook.Range["MyRange"];
DevExpress.Spreadsheet.Range myRange = worksheet["MyRange"];

Visual Basic

Dim myRange As DevExpress.Spreadsheet.Range = workbook.Range("MyRange")
Dim myRange As DevExpress.Spreadsheet.Range = worksheet("MyRange")

Defined Name Scope

Each defined name has a scope - an area (individual [worksheet](#) or entire [workbook](#)) where a name is recognized and can be used without qualification. For example, a defined name (*cellName*) whose scope is the first worksheet of a workbook (*Sheet1*) is recognized without qualification in this worksheet only (e.g., `=5+cellName`). To use this defined name in other worksheets, precede it with the name of the worksheet to which the defined name is scoped (e.g., `"=5+Sheet1!cellName"`). If the scope of a defined name (*cellName_global*) is an entire workbook, this name is recognized in any worksheet of this workbook (e.g., `"=5+cellName_global"`).

Each worksheet contained in a workbook, as well as the workbook itself, has its own collection of defined names (DefinedNameCollection) that can be accessed via the Worksheet.DefinedNames and [Workbook.DefinedNames](#) properties, respectively. Each name must be unique in its scope (use the DefinedNameCollection.Contains method to determine whether or not a specific name already exists in the collection). However, note that the same name can be used in different scopes.

When you use the defined name without preceding it by a worksheet name, this name is searched within the DefinedNameCollection collection of the worksheet where this name is used. Then, if the name is not found in the worksheet, it will be searched for in the workbook's collection of defined names. To use a global defined name explicitly, precede it with the workbook name (e.g., `"=5+WorkbookName.xlsx!cellName"`).

If the defined name is not found, the cell that uses this name displays the **#NAME?** [error](#).

Syntax Rules for Names

When creating and modifying defined names, follow the rules below.

- Start a name with a letter, the underscore symbol ("_") or the backslash ("\"). The remaining characters in the name can be letters, numbers, periods and underscore symbols.

Note that the single letters "C", "c", "R", or "r" cannot be used as defined names.

- A name cannot be the same as a cell reference (for example, "A1", "\$M\$15", etc.).
- A name cannot contain spaces (use underscore symbols and periods instead).
- A name cannot be an empty string.
- The length of a name cannot exceed 255 characters.
- Names are case-insensitive. For example, you are not allowed to create the *Products* and *PRODUCTS* names in one scope.

Create Defined Names

You can create defined names via the [Range.Name](#) property, or the [Worksheet.DefinedNames.Add](#) and [Workbook.DefinedNames.Add](#) methods.

- **Range.Name**

Access an object that specifies a [cell](#) or [cell range](#) to be named and set its Range.Name property. The corresponding DefinedName object is automatically created and added to the Worksheet.DefinedNames collection of the worksheet that contains the named cell or cell range. Thus, this worksheet is a [scope](#) of the created name. The DefinedName.RefersTo property is automatically set to the absolute cell reference (including the worksheet name).

C#

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
Worksheet sheet1 = workbook.Worksheets["Sheet1"];
Worksheet sheet2 = workbook.Worksheets["Sheet2"];
// Create a range.
Range range = sheet1.Range["A1:C3"];
// Specify the name for the created range.
range.Name = "namedRange";
// Access an object specifying the created name.
DefinedName definedName = sheet1.DefinedNames.GetDefinedName("namedRange");
// Use the defined name in the scope worksheet.
sheet1.Cells["D4"].Formula = "=SUM(namedRange)";
// Use the defined name in another worksheet.
sheet2.Cells["D4"].Formula = "=SUM(Sheet1!namedRange)";
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
Dim sheet1 As Worksheet = workbook.Worksheets("Sheet1")
Dim sheet2 As Worksheet = workbook.Worksheets("Sheet2")
' Create a range.
Dim range As Range = sheet1.Range("A1:C3")
' Specify the name for the created range.
range.Name = "namedRange"
' Access an object specifying the created name.
Dim definedName As DefinedName = sheet1.DefinedNames.GetDefinedName("namedRange")
' Use the defined name in the scope worksheet.
sheet1.Cells("D4").Formula = "=SUM(namedRange)"
' Use the defined name in another worksheet.
sheet2.Cells("D4").Formula = "=SUM(Sheet1!namedRange)"
```

• **Worksheet.DefinedNames.Add**

This method allows you to create a defined name whose scope is the specified worksheet and associate this name with an individual cell, range of cells, formula or constant value.

C#	
<pre>using DevExpress.Spreadsheet; // ... Workbook workbook = new Workbook(); Worksheet sheet1 = workbook.Worksheets["Sheet1"]; Worksheet sheet2 = workbook.Worksheets["Sheet2"]; // Access the "Sheet1" worksheet's collection of defined names. DefinedNameCollection sheet1_DefinedNames = sheet1.DefinedNames; // Create a defined name for a range of cells. sheet1_DefinedNames.Add("items", "Sheet1!\$A\$1:\$C\$3"); // Create a defined name for a formula. sheet1_DefinedNames.Add("totalSum", "=SUM(items)"); // Create a defined name for a constant. sheet1_DefinedNames.Add("coefficient", "=3"); // Use created names in the scope worksheet. sheet1.Cells["D4"].Formula = "=coefficient*totalSum"; // Use created names in another worksheet. sheet2.Cells["D4"].Formula = "=Sheet1!coefficient*Sheet1!totalSum";</pre>	

Visual Basic	
<pre>Imports DevExpress.Spreadsheet ' ... Dim workbook As New Workbook() Dim sheet1 As Worksheet = workbook.Worksheets("Sheet1") Dim sheet2 As Worksheet = workbook.Worksheets("Sheet2") ' Access the "Sheet1" worksheet's collection of defined names. Dim sheet1_DefinedNames As DefinedNameCollection = sheet1.DefinedNames ' Create a defined name for a range of cells. sheet1_DefinedNames.Add("items", "Sheet1!\$A\$1:\$C\$3") ' Create a defined name for a formula. sheet1_DefinedNames.Add("totalSum", "=SUM(items)") ' Create a defined name for a constant. sheet1_DefinedNames.Add("coefficient", "=3") ' Use created names in the scope worksheet. sheet1.Cells("D4").Formula = "=coefficient*totalSum" ' Use created names in another worksheet. sheet2.Cells("D4").Formula = "=Sheet1!coefficient*Sheet1!totalSum"</pre>	

• **Workbook.DefinedNames.Add**

This method allows you to create a defined name whose scope is the entire workbook and associate this name with an individual cell, range of cells, formula or constant value.

C#	
-----------	--

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
Worksheet sheet1 = workbook.Worksheets["Sheet1"];
Worksheet sheet2 = workbook.Worksheets["Sheet2"];
Worksheet sheet3 = workbook.Worksheets["Sheet3"];
// Create a defined name to be recognized in any worksheet of a workbook.
workbook.DefinedNames.Add("cellName_Global", "=Sheet2!$G$20");
// Use the defined name in different worksheets.
sheet1.Cells["A1"].Formula = "=cellName_Global";
sheet2.Cells["A1"].Formula = "=cellName_Global";
sheet3.Cells["A1"].Formula = "=cellName_Global";
```

Visual Basic
<pre>Imports DevExpress.Spreadsheet ' ... Dim workbook As New Workbook() Dim sheet1 As Worksheet = workbook.Worksheets("Sheet1") Dim sheet2 As Worksheet = workbook.Worksheets("Sheet2") Dim sheet3 As Worksheet = workbook.Worksheets("Sheet3") ' Create a defined name to be recognized in any worksheet of a workbook. workbook.DefinedNames.Add("cellName_Global", "=Sheet2!\$G\$20") ' Use the defined name in different worksheets. sheet1.Cells("A1").Formula = "=cellName_Global" sheet2.Cells("A1").Formula = "=cellName_Global" sheet3.Cells("A1").Formula = "=cellName_Global"</pre>

Access and Change Defined Names

All defined names of a spreadsheet document are stored in the `DefinedNameCollection` collections. Use the `Worksheet.DefinedNames` property to access the collection of defined names whose scope is a specific worksheet, or the [Workbook.DefinedNames](#) property to get defined names whose scope is an entire workbook. To obtain an individual defined name from the required collection, use the properties and methods of the corresponding `DefinedNameCollection` object. For example, you can get a defined name by its index in the collection or by its name (`DefinedNameCollection.GetDefinedName`).

An individual defined name is specified by the `DefinedName` object. Use this object's properties (`DefinedName.Name`, `DefinedName.RefersTo` and `DefinedName.Comment`) to modify the corresponding defined name as required.

Note
After you change an existing defined name, all instances of this name in a workbook will also be changed. For example, if you change DefinedName.Name , all cells using the old name will display the #NAME? error .

Delete Defined Names

To delete an existing defined name, use the `DefinedNameCollection.Remove` or `DefinedNameCollection.RemoveAt` method. To remove all defined names from the collection, use the `DefinedNameCollection.Clear` method.

Note
After you delete a name, all cells using that name will display the #NAME? error . After you delete a named cell or range of cells, all cells using defined names that refer to the deleted cell or cell range will display the #REF! error .

See Also

- [How to: Create a Named Range of Cells](#)
- [How to: Create Named Formulas](#)
- [How to: Use Names in Formulas](#)

Data Binding


[Office File API](#) > [Spreadsheet Document API](#) > [Data Binding](#)

This topic introduces the main concepts of data binding in the Spreadsheet Document API. You can bind a specific cell range in a worksheet to a data source or use a cell range as a data source for any DevExpress or third-party data-aware control (Data Grid, Chart Control, etc.). Data binding is two-way by default.

Worksheet data bindings are `WorksheetDataBinding` objects stored in the `WorksheetDataBindingCollection` collection, which can be accessed using the `Worksheet.DataBindings` property.

Bind a Cell Range to a Data Source

The Spreadsheet Document API provides the `WorksheetDataBindingCollection.BindToDataSource` method overloads that let you bind a specific cell range in a worksheet to a data source. The data source can be a [DataTable](#), [BindingList<T>](#) class instance, or an implementer of the [IBindingList](#), [ITypedList](#), or [IEnumerable](#) interface.

 **Important**

The Spreadsheet does not support hierarchical data sources, i.e., data sources with master-detail relationships set up between lists of objects.

If a worksheet range is bound to a read-only data source, the range itself is also read-only and its data cannot be updated. If a worksheet range is bound to a data source that allows changing notifications, such as a data source with the [IBindingList](#) interface, the worksheet range has a number of rows synchronized with the number of data records. The worksheet automatically inserts or deletes rows when the data source inserts or deletes its records. When a row is inserted or deleted in the bound range, the data record is inserted or deleted in the data source.

An `ExternalDataSourceOptions` class instance is passed to the `WorksheetDataBindingCollection.BindToDataSource` method and contains various options that allow you to control how data is retrieved from an external data source.

Option	Description
<code>DataSourceOptionsBase.CellValueConverter</code>	Allows you to specify a converter that converts data to cell values and back.
<code>ExternalDataSourceOptions.ImportHeaders</code>	Retrieves data field names and displays them in the first row of the specified worksheet range.
<code>DataSourceOptionsBase.SkipHiddenColumns</code>	Allows you to skip hidden columns while importing data from the data source.
<code>DataSourceOptionsBase.SkipHiddenRows</code>	Allows you to skip hidden rows while importing data from the data source.

Bind a Worksheet Table to a Data Source

To create a worksheet Table bound to the data source, use the `WorksheetDataBindingCollection.BindTableToDataSource` method. It returns the newly created worksheet table bound to the specified data source. The method also creates a `WorksheetTableDataBinding` object and adds it to the `Worksheet.DataBindings` collection.

The `WorksheetDataBindingCollection.BindTableToDataSource` method extends the `WorksheetDataBindingCollection.BindToDataSource` method, described earlier in this document, to worksheet tables. It uses the same parameters and returns an object that is the `WorksheetDataBinding` class descendant.

When the data binding is deleted, the table remains in the worksheet filled with data. When the table is deleted, the data binding is also deleted.

Use a Cell Range as a Data Source

To use a cell range as a data source for a data-aware control, create a data source object from the required cell range using the `WorksheetDataBindingCollection.CreateDataSource` method or the equivalent `Range.GetDataSource` method. The created data source object implements the [IBindingList](#), [ITypedList](#), [ICancelAddNew](#) and [IDisposable](#) interfaces and can be used to send/receive data to/from any data-aware control that supports the mentioned interfaces.

If `RangeDataSourceOptions.UseFirstRowAsHeader` is **false** and `RangeDataSourceOptions.DataSourceColumnTypeDetector` is not specified, field names are generated automatically (each field in the created data source has the default name *"Column 0"*,

"Column 1", etc.) and the field type is automatically determined based on the value and number format of the first existing cell of the corresponding column in the data source range.

A `RangeDataSourceOptions` class inherits from `DataSourceOptionsBase` and provides different options that allow you to control how data is extracted from the worksheet.

Option	Description
<code>DataSourceOptionsBase.CellValueConverter</code>	Allows you to specify a converter that converts cell values to custom objects and vice versa.
<code>DataSourceOptionsBase.SkipHiddenColumns</code>	Allows you to specify whether to include hidden columns into the resulting data source.
<code>DataSourceOptionsBase.SkipHiddenRows</code>	Allows you to specify whether to include hidden rows into the resulting data source.
<code>RangeDataSourceOptions.PreserveFormulas</code>	Allows you to restrict editing formulas in the data source range.
<code>RangeDataSourceOptions.EditingOptions</code>	Allows you to specify the editing options for the resulting data source.
<code>RangeDataSourceOptions.UseFirstRowAsHeader</code>	Specifies whether to use the text contained in the first row cells of the data source range as column headers.
<code>RangeDataSourceOptions.DataSourceColumnTypeDetector</code>	Allows you to explicitly specify the name and type of each column in the resulting data source.
<code>RangeDataSourceOptions.CellValueComparer</code>	Allows you to specify the comparer used to sort cell values in the data source columns.

Use a Worksheet Table as a Data Source

To use a worksheet Table as a data source for a data-aware control, create a data source object from the required cell range using the `Table.GetDataSource` method.

The method and its parameters are similar to the `Range.GetDataSource` method described previously in this document.

Examples

The following examples illustrate the data binding functionality of the Spreadsheet Document API.

- [How to: Use Worksheet Table as a Data Source](#)

See Also

[How to: Import Data to a Worksheet](#)

[How to: Use Worksheet Table as a Data Source](#)

[Office File API](#) > [Spreadsheet Document API](#) > [Pivot Tables](#)

A *pivot table* represents a summary table used to explore, analyze and aggregate huge amounts of data in a worksheet. It helps break your data into categories and subcategories, and automatically calculates subtotals and grand totals using the most suitable summary function from a predefined list.

	A	B	C	D	E	F
1	DATE	QUARTER	CUSTOMER	PRODUCT	CATEGORY	AMOUNT
2	1/1/2014	Q1	A&B Supermarkets	Wimmers gute Semmelknödel	Grains/Cereals	\$ 1,886.00
3	1/7/2014	Q1	Miller's	Mozzarella di Giovanni	Dairy Products	\$ 8,002.00
4	1/11/2014	Q1	Food Land	Mascarpone Fabioli	Dairy Products	\$ 6,392.00
5	1/11/2014	Q1	Food Land	Mozzarella di Giovanni	Dairy Products	\$ 4,850.00
6	1/25/2014	Q1	A&B Supermarkets	Camembert Pierrot	Dairy Products	\$ 6,444.00
7	2/10/2014	Q1	A&B Supermarkets	Gustaf's Knäckebröd	Grains/Cereals	\$ 1,768.00
8	3/8/2014	Q1	Big Foods	Mascarpone Fabioli	Dairy Products	\$ 8,652.00
9	3/8/2014	Q1	Big Foods	Gnocchi di nonna Alice	Grains/Cereals	\$ 3,950.00
10	3/8/2014	Q1	Food Land	Gnocchi di nonna Alice	Grains/Cereals	\$ 5,525.00
11	3/14/2014	Q1	A&B Supermarkets	Mascarpone Fabioli	Dairy Products	\$ 5,750.00
12	3/14/2014	Q1	Food Land	Singaporean Hokkien Fried Mee	Grains/Cereals	\$ 2,950.00
13	3/24/2014	Q1	Miller's	Camembert Pierrot	Dairy Products	\$ 5,980.00
14	3/24/2014	Q1	Miller's	Singaporean Hokkien Fried Mee	Grains/Cereals	\$ 5,100.00
15	4/9/2014	Q2	A&B Supermarkets	Mozzarella di Giovanni	Dairy Products	\$ 8,270.00
16	5/5/2014	Q2	A&B Supermarkets	Camembert Pierrot	Dairy Products	\$ 4,850.00
17	5/5/2014	Q2	A&B Supermarkets	Mascarpone Fabioli	Dairy Products	\$ 6,250.00

Source Data

	A	B	C	D	E	F
1						
2		CATEGORY	(All)			
3						
4		Products	Sum of AMOUNT			
5		Camembert Pierrot	\$ 63,392.00			
6		Gnocchi di nonna Alice	\$ 29,692.00			
7		Gustaf's Knäckebröd	\$ 19,159.00			
8		Mascarpone Fabioli	\$ 62,246.00			
9		Mozzarella di Giovanni	\$ 76,188.00			
10		Singaporean Hokkien Fried Mee	\$ 45,675.00			
11		Wimmers gute Semmelknödel	\$ 32,861.00			
12		Grand Total	\$ 329,213.00			
13						
14						

One-dimensional PivotTable Report


	A	B	C	D	E	F
1						
2	DATE	(All)				
3						
4	Sum of AMOUNT	Customers				
5	Products	A&B Supermarkets	Big Foods	Food Land	Miller's	Grand Total
6	Camembert Pierrot	\$ 21,600.00	\$ 12,872.00	\$ 13,146.00	\$ 15,774.00	\$ 63,392.00
7	Gnocchi di nonna Alice		\$ 11,488.00	\$ 18,204.00		\$ 29,692.00
8	Gustaf's Knäckebröd	\$ 8,986.00			\$ 10,173.00	\$ 19,159.00
9	Mascarpone Fabioli	\$ 19,300.00	\$ 14,524.00	\$ 14,502.00	\$ 13,920.00	\$ 62,246.00
10	Mozzarella di Giovanni	\$ 26,558.00	\$ 17,816.00	\$ 19,466.00	\$ 12,348.00	\$ 76,188.00
11	Singaporean Hokkien Fried Mee	\$ 17,756.00	\$ 9,383.00	\$ 7,920.00	\$ 10,616.00	\$ 45,675.00
12	Wimmers gute Semmelknödel	\$ 7,671.00		\$ 13,314.00	\$ 11,876.00	\$ 32,861.00
13	Grand Total	\$ 101,871.00	\$ 66,083.00	\$ 86,552.00	\$ 74,707.00	\$ 329,213.00
14						

Two-dimensional PivotTable Report

	A	B	C	D	E	F	G	
1		DATE	(All)					
2								
3		Sum of AMOUNT	Quarters					
4		Products	Q1	Q2	Q3	Q4	Grand Total	
5		A&B Supermarkets	\$ 15,848.00	\$ 29,681.00	\$ 24,094.00	\$ 32,248.00	\$ 101,871.00	
6		Dairy Products	\$ 12,194.00	\$ 22,692.00	\$ 9,656.00	\$ 22,916.00	\$ 67,458.00	
7		Grains/Cereals	\$ 3,654.00	\$ 6,989.00	\$ 14,438.00	\$ 9,332.00	\$ 34,413.00	
8		Gustaf's Knäckebröd	\$ 1,768.00		\$ 4,098.00	\$ 3,120.00	\$ 8,986.00	
9		Singaporean Hokkien Fried Mee		\$ 4,444.00	\$ 7,100.00	\$ 6,212.00	\$ 17,756.00	
10		Wimmers gute Semmelknöde	\$ 1,886.00	\$ 2,545.00	\$ 3,240.00		\$ 7,671.00	
11		Big Foods	\$ 12,602.00	\$ 16,565.00	\$ 22,325.00	\$ 14,591.00	\$ 66,083.00	
12		Dairy Products	\$ 8,652.00	\$ 8,965.00	\$ 17,950.00	\$ 9,645.00	\$ 45,212.00	
13		Grains/Cereals	\$ 3,950.00	\$ 7,600.00	\$ 4,375.00	\$ 4,946.00	\$ 20,871.00	
14		Gnocchi di nonna Alice	\$ 3,950.00	\$ 2,592.00		\$ 4,946.00	\$ 11,488.00	
15		Singaporean Hokkien Fried Mee		\$ 5,008.00	\$ 4,375.00		\$ 9,383.00	

Multi-dimensional PivotTable Report

All pivot tables in a worksheet are stored in the PivotTableCollection object, which you can access using the Worksheet.PivotTables property. The PivotTableCollection interface provides the basic methods of working with pivot tables in code. Use the PivotTableCollection.Add method to create a pivot table based on the cell range in a source worksheet or using the data cache of the existing PivotTable report (PivotCache). For an example on how to insert a pivot table, refer to the [How to: Create a Pivot Table](#) article.

 Important	
Currently, you can use only worksheet data as a data source for a pivot table. External data sources (such as ODC files, OLAP cubes, relational databases, XML files, etc.) are not supported.	

An individual pivot table is represented by the PivotTable object and can be accessed by its name in the pivot table collection. You can change the source data for your pivot table or move it to a new location in a worksheet by using the PivotTable.ChangeDataSource and PivotTable.MoveTo methods, respectively.

PivotTable Structure

After you insert and position a pivot table in a worksheet, you should fill it with data by adding necessary [fields](#) (columns of the source range) to the report. All fields are stored in the PivotFieldCollection accessible from the PivotTable.Fields property. To add a field to the PivotTable report, access this field by its name in the collection (by default, the label of the corresponding column is used as the field name) and move it to the required PivotTable [area](#):

- **row axis area** - represented by the collection of [row fields](#) accessible from the PivotTable.RowFields property;
- **column axis area** - represented by the collection of [column fields](#) accessible from the PivotTable.ColumnFields property;
- **page/report filter area** - represented by the collection of [page fields](#) accessible from the PivotTable.PageFields property;
- **data area** - represented by the collection of [data fields](#) accessible from the PivotTable.DataFields property.

Subsequently, you can move the desired field to another area of the pivot table to change the report layout, or you can re-order fields in a specific area using the **MoveDown**, **MoveUp**, **MoveToBeginning** or **MoveToEnd** method called for the field whose position you wish to change. To remove a field from the pivot table, use the **Remove** or **RemoveAt** method of the collection containing this field.

In addition to fields from the source data, the pivot table may also contain *calculated fields*, whose values are produced based on custom formulas. To add a calculated field to the PivotTable report, use the PivotCalculatedFieldCollection.Add method of the PivotTable.CalculatedFields collection.

Each non-calculated field is made up of *items* - unique data entries contained in the field. All field items are stored in the PivotItemCollection collection accessible using the PivotField.Items property. You can sort items within a field (PivotField.SortType, PivotField.SortItems), filter them (PivotTable.Filters), or group them to create new subsets of data (PivotField.GroupItems).

A pivot field may also contain one or more *calculated items*. To add a calculated item to the PivotTable field, use the PivotCalculatedItemCollection.Add method of the PivotField.CalculatedItems collection.

PivotTable Settings and Capabilities

After you create a pivot table and populate it with data, you can modify its settings to improve the readability and comprehension of your report. You can adjust the report layout, apply a built-in or custom style to the pivot table, filter field items to display only significant values, and more. To do this, use properties of the PivotTable object listed in the table below.

Property	Description
PivotTable.Layout	Provides access to the layout settings of the pivot table that enable you to show the report in a compact, outline or tabular form, display or hide subtotals and grand totals, insert a blank row after each group of items, etc.
PivotTable.Style	Allows you to set the predefined or custom style to be applied to the pivot table.
PivotTable.BandedColumns PivotTable.BandedRows	Allow you to apply the banded column/row formatting to the pivot table.
PivotTable.ShowColumnHeaders PivotTable.ShowRowHeaders	Allow you to format column and row headers in the PivotTable report.
PivotTable.View	Provides access to the display options of the pivot table.
PivotTable.Filters	Allows you to specify filtering criteria to show specific values in the pivot table.
PivotTable.Behavior	Allows you to apply restrictions on different PivotTable operations.

See Also

- [Pivot Table Structure](#)
- [Pivot Table Examples](#)

Pivot Table Structure

[Office File API](#) > [Spreadsheet Document API](#) > [Pivot Tables](#) > [Pivot Table Structure](#)

This topic provides general information about the structure of a [PivotTable report](#).

A pivot table refers to each column in the source [range](#) as a **field**. Each field is made up of **items** - unique data entries contained in this field.

In the initial stage, when you add a pivot table to a worksheet, it contains no data. To populate the created report with data, you should place the available data source fields into one of the four PivotTable sections.

Page/Filter Field

	A	B	C	D	E	
1						
2		CATEGORY	(All)			
3						
4		Sum of AMOUNT	Customer			
5		Product	A&B Supermarkets	Big Foods	Food Land	
6		Camembert Pierrot	\$ 21,600.00	\$ 12,872.00	\$ 13,146.00	
7		Gnocchi di nonna Alice	\$ 13,500.00	\$ 11,488.00	\$ 18,204.00	
8		Gustaf's Knäckebröd	\$ 8,986.00	\$ 12,865.00	\$ 10,540.00	
9		Mascarpone Fabioli	\$ 19,300.00	\$ 14,524.00	\$ 14,502.00	
10		Mozzarella di Giovanni	\$ 26,558.00	\$ 17,816.00	\$ 19,466.00	
11		Singaporean Hokkien Fried Mee	\$ 17,756.00	\$ 9,383.00	\$ 7,920.00	
12		Wimmers gute Semmelknödel	\$ 7,671.00	\$ 8,530.00	\$ 13,314.00	
13						

Column Field

Data Field

Row Field

Page/Report Filter Area

Row Axis Area

Column Axis Area

Data Area

• Row Axis Area/Row Area

Contains fields used to group report data by rows. A pivot table containing multiple row fields has one **inner row field** (the one located closest to the data area). All other row fields are regarded as **outer row fields**. Items of the outermost row field are displayed only once, while items in the rest of the row fields can be repeated.

Inner Row Field

'PRODUCT' Field Items

	A	B	C	D	E
1					
2		Sum of AMOUNT	CUSTOMER		
3		CATEGORY	PRODUCT	A&B Supermarkets	Big Foods
4			Camembert Pierrot	\$ 21,600.00	\$ 12,872.00
5		Dairy Products	Mascarpone Fabioli	\$ 19,300.00	\$ 14,524.00
6			Mozzarella di Giovanni	\$ 26,558.00	\$ 17,816.00
7		Dairy Products Total		\$ 67,458.00	\$ 45,212.00
8			Gnocchi di nonna Alice		\$ 11,488.00
9		Grains/Cereals	Gustaf's Knäckebröd	\$ 8,986.00	\$ 8,986.00
10			Singaporean Hokkien Fried Mee	\$ 17,756.00	\$ 9,383.00
11			Wimmers gute Semmelknödel	\$ 7,671.00	\$ 7,671.00
12		Grains/Cereals Total		\$ 34,413.00	\$ 20,871.00
13		Grand Total		\$ 101,871.00	\$ 66,083.00
14					

Outer Row Field

'CATEGORY' Field Items

• Column Axis Area/Column Area

Contains fields used to break report data into categories by columns. A pivot table containing multiple column fields has one **inner column field**. All other column fields are regarded as **outer column fields**. Items of the outermost column field are displayed only once, while items in the rest of the column fields can be repeated.

	A	B	C	D	E
1					
2	Sum of AMOUNT	YEAR	QUARTER		
3		2014	2015		
4	PRODUCT	Q3	Q4	Q3	Q4
5	Camembert Pierrot	\$ 8,220.00	\$ 12,708.00	\$ 6,944.00	\$ 7,100.00
6	Gnocchi di nonna Alice	\$ 4,125.00			\$ 13,500.00
7	Gustaf's Knäckebröd		\$ 8,558.00	\$ 4,098.00	
8	Mascarpone Fabioli	\$ 2,394.00	\$ 5,716.00	\$ 5,872.00	\$ 15,230.00
9	Mozzarella di Giovanni	\$ 8,026.00	\$ 14,283.00	\$ 9,834.00	\$ 5,450.00
10	Singaporean Hokkien Fried Mee	\$ 9,891.00		\$ 8,370.00	\$ 6,212.00
11	Wimmers gute Semmelknödel	\$ 10,780.00	\$ 6,230.00	\$ 3,240.00	\$ 4,200.00
12	Grand Total	\$ 43,436.00	\$ 47,495.00	\$ 38,358.00	\$ 51,692.00
13					

• Data Area

Contains fields against which summaries are calculated. As a rule, data fields contain numeric values, which are summarized with the *Sum* function, but you can change the calculation type by selecting one of the predefined functions (*Count*, *Average*, *Min*, *Max*, *Product*, *Count Numbers*, *StdDev*, *StdDevp*, *Var*, or *Varp*). For data fields containing textual values, the *Count* summary function is used by default.

• Page/Report Filter Area

Contains fields used to filter the entire PivotTable report to display data for the selected items.

For details on how to use the **Pivot Table API** to create a pivot table and organize its structure, refer to the [How to: Create a Pivot Table](#) example.

Mail Merge

[Office File API](#) > [Spreadsheet Document API](#) > [Mail Merge](#)

This section contains the following topics:

- [Mail Merge Overview](#)
- [Template Document](#)
- [Mail Merge Functions](#)

Mail Merge Overview

[Office File API](#) > [Spreadsheet Document API](#) > [Mail Merge](#) > [Mail Merge Overview](#)

The **Mail Merge** functionality enables you to automatically generate a set of documents based on a single template and include unique data values retrieved from a data source into each document. This feature can be useful for a variety of business requirements, such as personalizing letters, and composing catalogs and reports.

Mail Merge Process

To perform a mail merge, you need a **template** and a **data source**.

- A template is a document containing placeholders for the information that will be merged from a data source ([mail merge fields](#)). See the [Template Document](#) topic to learn more about mail merge templates.
- A data source contains data that will be merged into fields in a template to create merged documents. A data source can be any object that exposes the **IList** interface, such as a [ArrayList](#) or a [DataTable](#). So, you can create a mail merge data source at runtime, or retrieve data from an external database using data adapters.

The data source is bound to the template using the [Workbook.MailMergeDataSource](#) and [Workbook.MailMergeDataMember](#) properties of the template workbook.

Use the template workbook's [Workbook.MailMergeOptions](#) property to get access to the mail merge options.

The template workbook's [Workbook.GenerateMailMergeDocuments](#) method finalizes the mail merge process. It returns a collection of resulting workbooks (if the Single Sheet or Multiple Sheets [mail merge mode](#) is used, the collection will contain a single workbook). You can open the resulting workbook or save it to a file or stream.

See Also

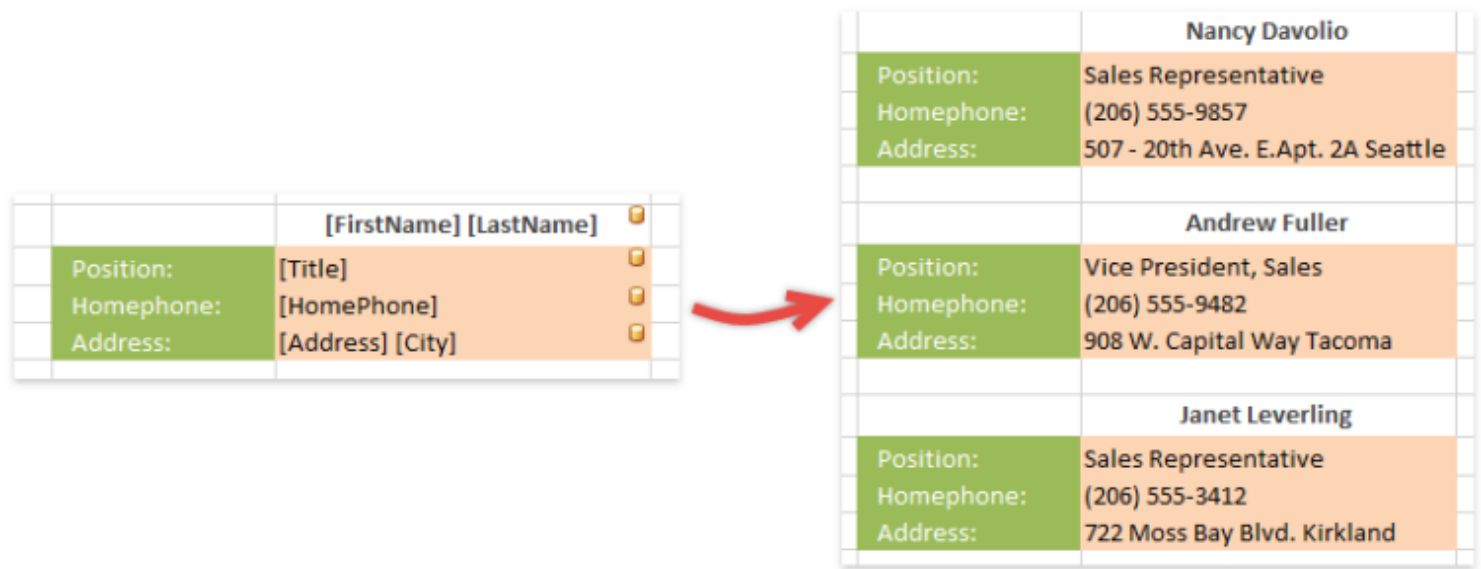
[How to: Perform a Mail Merge](#)

Template Document

[Office File API](#) > [Spreadsheet Document API](#) > [Mail Merge](#) > [Template Document](#)

Creating a template is one of the steps required to perform a mail merge with [Workbook](#) - merged documents are generated based on a template ([Workbook.GenerateMailMergeDocuments](#)) bound to a data source ([Workbook.MailMergeDataSource](#) and [Workbook.MailMergeDataMember](#)).

A mail merge template is a [workbook](#) with a single [worksheet](#). It contains the information that will be the same in each merged document, as well as the placeholders ([mail merge fields](#)) that will be filled with unique data values retrieved from the bound data source in each merged document.



A template also holds special [defined names](#) indicating whether data records should be merged into a single worksheet, separate worksheets or separate documents ([mail merge mode](#)), and whether the resulting worksheet should have a vertical or horizontal orientation (the [document orientation](#)).

Note

Loading the mail merge templates with the `ObjectDataSource` data source may be unsafe if the data source is contained in a compiled assembly. Create and register a custom service that implements the `IObjectDataSourceValidationService` interface to validate an `ObjectDataSource` contained in the template and prevent the data source from loading.

This topic explains how to create a mail merge template and describes its elements.

- [Create a Template](#)
- [Insert Fields](#)
- [Specify Template Ranges](#)
- [Select a Mail Merge Mode](#)
- [Set Document Orientation](#)

Create a Template

The easiest way to create a mail merge template is to load the template document into the `SpreadsheetControl` and modify it using the `SpreadsheetControl`'s Mail Merge Ribbon UI.

Alternatively, you can create a template worksheet programmatically. Provide static content, insert the required [mail merge fields](#), specify [template ranges](#) and select the [mail merge mode](#) and [document orientation](#).

Insert Fields

A *mail merge field* is a placeholder in which a value from the specified data source field will appear in a merged document. When you perform a mail merge, a separate merged document (depending on the [mail merge mode](#) used, it can be a cell range, worksheet or workbook) is created for each record of the bound data source. Each merged document is a template copy where mail merge fields are replaced with unique information from a particular row of the data source.

To insert fields into template cells, use the following special functions in [cell formulas](#): FIELD and FIELDPICTURE. Specify the name of the data source field from which values should be inserted in place of a merge field as a function argument. For more information on these functions, see the [Mail Merge Functions](#) document.

Note

You need to know the column names of the data source to insert mail merge fields.

Specify Template Ranges

Besides an ordinary mail merge template that is simply copied for each record of the bound data source, you can also create an advanced template by specifying **Detail**, **Header** and **Footer** ranges.

- Detail**
When you specify a detail range in a template, only this range (with header and footer ranges, if they are also specified) will be copied for each record of the bound data source and will appear in the resulting document. [Mail merge fields](#) are usually placed within the detail range.
- Header and Footer**
Header and footer ranges can be specified in addition to a detail range. These ranges are placed above and below (if the vertical [document orientation](#) is used) all detail range instances in the resulting document if all data records are merged into a single worksheet. In other [mail merge modes](#), the header and footer accompany each detail range copy. Header and footer ranges do not take effect if a detail range is not specified in the template.

Since a template range is actually a cell range specified by the corresponding [defined name](#) ("DETAILRANGE", "HEADERRANGE" and "FOOTERRANGE"), you can programmatically divide a template into ranges by naming cell ranges. For example, the following code creates the same detail and header ranges in the template as shown in the image above.

```
C#
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new DevExpress.Spreadsheet.Workbook();
Worksheet template = workbook.Worksheets[0];
// Set a detail range in the template.
Range detail = template.Range["A6:E10"];
detail.Name = "DETAILRANGE";
// Set a header range in the template.
Range header = template.Range["A1:E2"];
header.Name = "HEADERRANGE";
```

```
Visual Basic
Imports DevExpress.Spreadsheet
' ...
Dim workbook As Workbook = New DevExpress.Spreadsheet.Workbook()
Dim template As Worksheet = workbook.Worksheets(0)
' Set a detail range in the template.
Dim detail As Range = template.Range("A6:E10")
detail.Name = "DETAILRANGE"
' Set a header range in the template.
Dim header As Range = template.Range("A1:E2")
header.Name = "HEADERRANGE"
```

Defined names specifying template ranges are added to the Worksheet.DefinedNames collection of the template worksheet.

Mail Merge Mode

When creating a template, you can specify how the result of the mail merge should be generated by selecting one of the available modes.

- Multiple Documents** - The merged range for each record of the data source is inserted into a separate workbook.
- Multiple Sheets** - The merged range for each record of the data source is inserted into a separate worksheet in a single workbook.
- Single Sheet** - Merged ranges for all records of the data source are inserted one after the other (vertically or horizontally, depending on the [document orientation](#)) into a single worksheet. If a template is divided into [ranges](#), the resulting worksheet will have the following structure: a header, a set of detail ranges repeated for all records of the data source, and a footer. This mode is used by default.

The selected mail merge mode is stored as a [defined name](#) in a mail merge template. By default, this defined name does not exist in the template workbook, and the "Single Sheet" mode is used. You can specify a mail merge mode programmatically by setting the "MAILMERGEMODE" defined name to the "Worksheets" or "OneWorksheet" string constant explicitly within the [Workbook.DefinedNames](#) collection of the template workbook.

```
C#
DevExpress.Spreadsheet.Workbook workbook = new DevExpress.Spreadsheet.Workbook();
// Select the "Multiple Documents" mail merge mode.
workbook.DefinedNames.Add("MAILMERGEMODE", "\"Documents\"");
// Switch the mail merge mode to "Multiple Sheets".
workbook.DefinedNames.GetDefinedName("MAILMERGEMODE").RefersTo = "\"Worksheets\"";
// Switch the mail merge mode to "Single Sheet".
workbook.DefinedNames.GetDefinedName("MAILMERGEMODE").RefersTo = "\"OneWorksheet\"";
```

```
Visual Basic
Dim workbook As New DevExpress.Spreadsheet.Workbook()
' Select the "Multiple Documents" mail merge mode.
workbook.DefinedNames.Add("MAILMERGEMODE", "\"Documents\"")
' Switch the mail merge mode to "Multiple Sheets".
workbook.DefinedNames.GetDefinedName("MAILMERGEMODE").RefersTo = "\"Worksheets\""
' Switch the mail merge mode to "Single Sheet".
workbook.DefinedNames.GetDefinedName("MAILMERGEMODE").RefersTo = "\"OneWorksheet\""
```

Document Orientation

Set the *document orientation* to specify how [template ranges](#) should be arranged in the resulting worksheet. Header, detail and footer ranges are located one after the other from top to bottom, or from left to right, depending on the document orientation selected.

In the ["Single Sheet" mode](#), detail range copies created for each record of the data source are placed one under the other (in a vertical orientation), or one to the right of the other (in a horizontal orientation) between a header and a footer.

The document orientation is saved as a [defined name](#) in the [Workbook.DefinedNames](#) collection of the template workbook. By default, this defined name does not exist and the vertical document orientation is used.

Document Orientation	Name	RefersTo
Vertical	HORIZONTALMODE	FALSE
Horizontal	HORIZONTALMODE	TRUE

See Also
[How to: Validate the ObjectDataSource Contained in the Spreadsheet MailMerge Template](#)

Target_range is a reference to the cell range in which a picture should be inserted.

Ignore_aspect_ratio is an optional Boolean parameter that indicates whether a picture's original aspect ratio should be ignored. If this parameter is not specified or set to FALSE, the picture's aspect ratio is retained.

OffsetX and OffsetY are values that specify the distance from the left and top of the target range in pixels. These parameters take effect only if *picture_placement* is set to "range" and *ignore_aspect_ratio* is set to FALSE.

Width and height are values that specify the desired width and height of the picture in pixels.

Result: In a mail merge template, this function inserts the name of the specified data field enclosed in square brackets: [data_field_name]. After a mail merge is complete, field images appear in the resulting document and the function returns an empty string. If the function parameters are incorrect or an image does not exist in the data source, the default string that is the field name in square brackets returns.

Example	Description
"=FIELDPICTURE("Picture", "range", A1:B2)" "=FIELDPICTURE("Picture", "range", A1:B2, FALSE)"	Inserts a picture and scales it to fit in the specified range of cells, locking the aspect ratio.
"=FIELDPICTURE("Picture", "range", A1:B2, TRUE)"	Inserts a picture to fit in the specified range of cells without locking the aspect ratio.
"=FIELDPICTURE("Picture", "topleft", A1)" "=FIELDPICTURE("Picture", "topleft", A1:B2)"	Inserts a picture so that its top left corner is located at the specified cell or at the top left cell of the specified range.
"=FIELDPICTURE("Picture", "topleft", A1, 100)" "=FIELDPICTURE("Picture", "topleft", A1:B2, 100)"	Inserts a picture so that its top left corner is located at the specified cell or at the top left cell of the specified range, and sets the image width to 100 pixels without retaining the aspect ratio.
"=FIELDPICTURE("Picture", "topleft", A1, 0, 100)" "=FIELDPICTURE("Picture", "topleft", A1:B2, 0, 100)"	Inserts a picture so that its top left corner is located at the specified cell or at the top left cell of the specified range, and sets the image height to 100 pixels without retaining the aspect ratio.
"=FIELDPICTURE("Picture", "topleft", A1, 120, 100)" "=FIELDPICTURE("Picture", "topleft", A1:B2, 120, 100)"	Inserts a picture so that its top left corner is located at the specified cell or at the top left cell of the specified range, and sets the image width to 120 pixels and height to 100 pixels without retaining the aspect ratio.

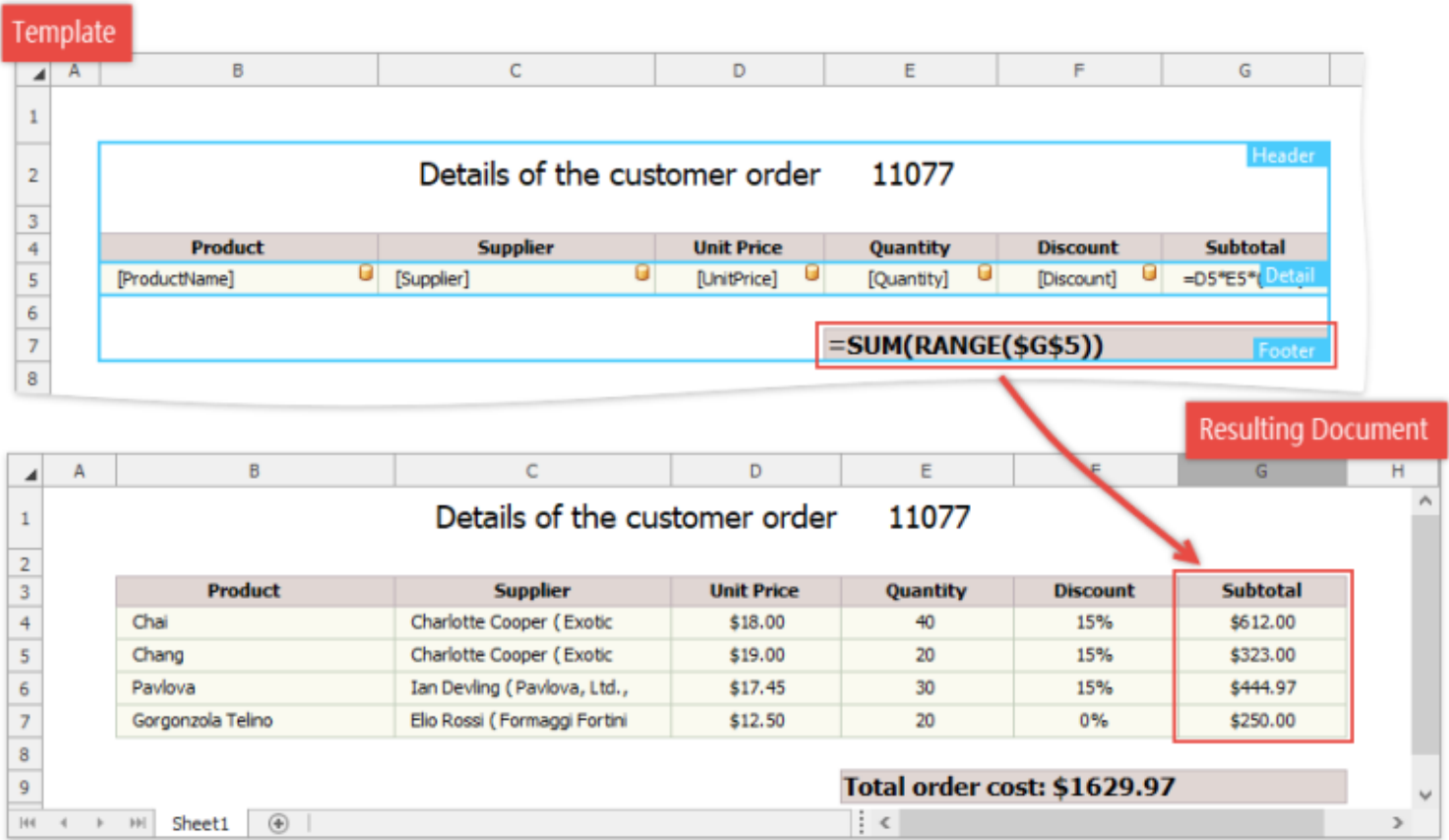
• RANGE

At the template creation stage of the mail merge, you can access the range into which the specified template cell will be expanded in a merged document after a template is repeated for each data record, and inserted one under the other in a single worksheet. To do this, use the RANGE function.

RANGE(abs_cell_reference)

Abs_cell_reference is an absolute reference to a cell in a template that will be copied for each data source record in the resulting worksheet.

Result: In a mail merge template, this function returns a value of the cell used as its parameter. In the resulting document, this function returns a cell range into which the template cell was expanded. If an error occurs when the target cell range is retrieved (e.g., when the specified cell reference points to a range that doesn't belong to a detail, header or footer range), the RANGE function returns the ParameterValue.InvalidValueInFunction value.



• **PARAMETER**

The **PARAMETER** function retrieves a mail merge parameter value and inserts it into the merged document.

PARAMETER("parameter_name")

Parameter_name is the name of the parameter whose value should be embedded into the resulting document.

Result: In a mail merge template, this function inserts the parameter name with the "Parameters." prefix enclosed in square brackets: [Parameters.parameter_name]. If the parameter value cannot be retrieved (e.g., when the parameter name is incorrect), this function returns the default string that is [Parameters.parameter_name]. Otherwise, the parameter value is returned.

Examples

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#)

This section provides a full list of examples (grouped by features) contained in this help.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

File Operations

- [How to: Load a Document to a Workbook](#)
- [How to: Save a Document to a File](#)
- [How to: Export a Workbook to PDF](#)
- [How to: Export a Document to HTML](#)

Workbooks

- [How to: Create a New Workbook](#)
- [How to: Clone a Workbook](#)
- [How to: Specify Document Properties](#)
- [How to: Merge Multiple Workbooks Into One Document](#)

Worksheets

- [How to: Access a Worksheet](#)
- [How to: Set an Active Worksheet](#)
- [How to: Add a New Worksheet](#)
- [How to: Delete a Worksheet](#)
- [How to: Rename a Worksheet](#)
- [How to: Copy Worksheets within a Workbook](#)
- [How to: Copy Worksheets between Workbooks](#)
- [How to: Move a Worksheet to another Location](#)
- [How to: Show and Hide a Worksheet](#)
- [How to: Show and Hide Gridlines](#)
- [How to: Set Page Orientation](#)
- [How to: Set Page Margins](#)
- [How to: Set Paper Size](#)
- [How to: Zoom In and Out of a Worksheet](#)

Rows and Columns

- [How to: Access a Row or Column](#)
- [How to: Add a New Row or Column to a Worksheet](#)
- [How to: Delete a Row or Column from a Worksheet](#)
- [How to: Copy a Row or Column](#)
- [How to: Hide a Row or a Column](#)
- [How to: Specify Row Height or Column Width](#)

Cells

- [How to: Access a Cell in a Worksheet](#)
- [How to: Access a Range of Cells](#)
- [How to: Insert a Cell or Cell Range](#)
- [How to: Delete a Cell or Range of Cells](#)
- [How to: Create a Named Range of Cells](#)

- [How to: Change a Cell or Cell Range Value](#)
- [How to: Add Formulas to Cells](#)
- [How to: Add a Hyperlink to a Cell](#)
- [How To: Add a Comment To a Cell](#)
- [How to: Clear Cells of Content, Formatting, Hyperlinks and Comments](#)
- [How to: Copy Cell Data Only, Cell Style Only, or Cell Data with Style](#)
- [How to: Merge Cells or Split Merged Cells](#)

Formulas

- [How to: Use Constants and Calculation Operators in Formulas](#)
- [How to: Use Cell and Worksheet References in Formulas](#)
- [How to: Use Names in Formulas](#)
- [How to: Create Named Formulas](#)
- [How to: Use Functions and Nested Functions in Formulas](#)
- [How to: Create Shared Formulas](#)
- [How to: Create Array Formulas](#)

Import and Export Data

- [How to: Import Data to a Worksheet](#)
- [How to: Export a Worksheet Range to a DataTable](#)

Formatting

- [How to: Apply a Style to a Cell or Range of Cells](#)
- [How to: Create or Modify a Style](#)
- [How to: Format a Cell or Range of Cells](#)
- [How to: Specify Number or Date Format for Cell Content](#)
- [How to: Change Cell Font and Background Color](#)
- [How to: Configure Cell Font Settings](#)
- [How to: Align Cell Content](#)
- [How to: Add and Remove Cell Borders](#)
- [How to: Clear Cell Formatting](#)

Conditional Formatting

- [How to: Format Cell Values that are Above or Below the Average](#)
- [How to: Format Cells that are Between or Not Between Two Values](#)
- [How to: Format Top or Bottom Ranked Values](#)
- [How to: Format Cells based on the Text in the Cell](#)
- [How to: Format Unique or Duplicate Values, Blank Cells and Formula Errors](#)
- [How to: Format Cells that Contain Dates](#)
- [How to: Format Cells that are Less than, Greater than or Equal to a Value](#)
- [How to: Use a Formula to Determine which Cells to Format](#)
- [How to: Format Cells Using a Two-Color Scale](#)
- [How to: Format Cells Using a Three-Color Scale](#)
- [How to: Format Cells Using Data Bars](#)
- [How to: Format Cells Using Icon Sets](#)

Tables

- [How to: Create a Table](#)
- [How to: Perform Calculations in a Table](#)
- [How to: Apply a Table Style](#)
- [How to: Create, Modify And Delete Table Styles](#)

Sort Data

- [How to: Sort Data in a Worksheet Range](#)

Filter Data

- [How to: Enable Filtering](#)
- [How to: Filter by Cell Values](#)
- [How to: Filter by Date Values](#)
- [How to: Apply a Custom Date Filter](#)
- [How to: Apply a Custom Text Filter](#)
- [How to: Apply a Custom Number Filter](#)
- [How to: Apply a Dynamic Filter](#)
- [How to: Filter Top or Bottom Ranked Values](#)
- [How to: Sort Data in the Filtered Range](#)
- [How to: Reapply a Filter](#)
- [How to: Clear a Filter](#)

Group Data

- [How to: Outline Data Manually](#)
- [How to: Outline Data Automatically](#)
- [How to: Insert Subtotals in a Data Range](#)

Find and Replace

- [How to: Perform Search in a Document](#)

Charts

- [How to: Create a Basic Chart](#)
- [How to: Format Chart Elements](#)
- [How to: Display the Chart Title](#)
- [How to: Display and Format Data Label](#)
- [How to: Show or Hide the Chart Legend](#)
- [How to: Change the Display of Chart Axes](#)
- [How to: Protect a Chart](#)

Pivot Tables

- [How to: Create a Pivot Table](#)
- [How to: Refresh a Pivot Table](#)
- [How to: Change a Data Source for a Pivot Table](#)
- [How to: Move a Pivot Table](#)
- [How to: Clear or Remove a Pivot Table](#)
- [How to: Change the PivotTable Layout](#)
- [How to: Subtotal Fields in a Pivot Table](#)
- [How to: Display or Hide Grand Totals for a Pivot Table](#)
- [How to: Apply a Predefined Style to a Pivot Table](#)
- [How to: Apply a Custom Style to a Pivot Table](#)
- [How to: Control Style Options](#)
- [How to: Change the Summary Function for a Data Field](#)
- [How to: Apply a Custom Calculation to a Data Field](#)
- [How to: Create a Calculated Field](#)
- [How to: Create a Calculated Item](#)
- [How to: Sort Items in a Pivot Table](#)
- [How to: Filter Items in a Pivot Table](#)

- [How to: Group Items in a Pivot Table](#)

Sparklines

- [How to: Create Sparklines](#)
- [How to: Group and Ungroup Sparklines](#)
- [How to: Customize the Sparkline Appearance](#)
- [How to: Specify Sparkline Axis Settings](#)
- [How to: Delete Sparklines](#)

Pictures

- [How to: Insert and Delete Pictures](#)
- [How to: Modify an Embedded Picture](#)

Protection

- [How to: Protect a Workbook](#)
- [How to: Protect a Worksheet](#)
- [How to: Protect Specific Worksheet Ranges](#)

Printing

- [How to: Print a Workbook](#)
- [How to: Specify Print Settings](#)
- [How to: Show a Print Preview Form for a Workbook](#)
- [How to: Add Headers and Footers to a Worksheet Printout](#)
- [How to: Define a Print Area](#)
- [How to: Print Titles on a Worksheet](#)
- [How to: Use the WPF Chart Rendering Mechanism When Printing or Exporting a Workbook to PDF](#)

Files

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Files](#)

This section contains the following examples:

- [How to: Load a Document to a Workbook](#)
- [How to: Save a Document to a File](#)
- [How to: Export a Workbook to PDF](#)
- [How to: Export a Document to HTML](#)

How to: Load a Document to a Workbook

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Files](#) > [How to: Load a Document to a Workbook](#)

Important

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

To load an existing spreadsheet document into the [Workbook](#) object, use the [Workbook.LoadDocument](#) method.

• Load from File

Create a new [Workbook](#) object and call the [Workbook.LoadDocument](#) method with the passed file path to load a workbook from the existing file. Specify the file format as the second parameter of the method using the `DocumentFormat` enumerator.

```
C#

// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
// Load a workbook from the file.
workbook.LoadDocument("Documents\\Document.xlsx", DocumentFormat.Xlsx);
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
' ...
Private workbook As New Workbook()
' Load a workbook from the file.
workbook.LoadDocument("Documents\\Document.xlsx", DocumentFormat.Xlsx)
```

• Load from Stream

Create the **FileStream** object with the specified file path to open the existing file, and call the [Workbook.LoadDocument](#) method with this stream object passed as a parameter. Specify the file format as the second parameter of the method using the `DocumentFormat` enumerator.

```
C#

// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
using System.IO;
// ...
Workbook workbook = new Workbook();
// Load a workbook from the stream.
using (FileStream stream = new FileStream("Documents\\Document.xlsx", FileMode.Open)) {
    workbook.LoadDocument(stream, DocumentFormat.Xlsx);
}
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
Imports System.IO
' ...
Dim workbook As New Workbook()
' Load a workbook from the stream.
Using stream As New FileStream("Documents\\Document.xlsx", FileMode.Open)
    workbook.LoadDocument(stream, DocumentFormat.Xlsx)
End Using
```

See Also

[Supported Formats](#)

[How to: Save a Document to a File](#)

How to: Save a Document to a File

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Files](#) > [How to: Save a Document to a File](#)

Important

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

To save a spreadsheet document loaded to the [Workbook](#) object, use the [Workbook.SaveDocument](#) method.

• Save to File

Call the [Workbook.SaveDocument](#) method with the passed file path to save a workbook to the file. Specify the file format as the second parameter of the method using the DocumentFormat enumerator.

```
C#

// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
// ...
// Save the modified document to the file.
workbook.SaveDocument("Documents\\SavedDocument.xlsx", DocumentFormat.Xlsx);
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
' ...
Private workbook As New Workbook()
' ...
' Save the modified document to the file.
workbook.SaveDocument("Documents\\SavedDocument.xlsx", DocumentFormat.Xlsx)
```

• Save to Stream

Create the **FileStream** object with the specified file path to save a workbook, and call the [Workbook.SaveDocument](#) method with this stream passed as a parameter. Specify the file format as the second parameter of the method using the DocumentFormat enumerator.

```
C#

// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
using System.IO;
// ...
Workbook workbook = new Workbook();
// ...
// Save the modified document to the stream.
using (FileStream stream = new FileStream("Documents\\SavedDocument.xlsx",
    FileMode.Create, FileAccess.ReadWrite)) {
    workbook.SaveDocument(stream, DocumentFormat.Xlsx);
}
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
Imports System.IO
' ...
Dim workbook As New Workbook()
' ...
' Save the modified document to the stream.
Using stream As New FileStream("Documents\\SavedDocument1.xlsx", FileMode.Create, FileAccess.ReadWrite)
    workbook.SaveDocument(stream, DocumentFormat.Xlsx)
End Using
```


See Also

[Supported Formats](#)


[How to: Load a Document to a Workbook](#)

How to: Export a Workbook to PDF

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Files](#) > [How to: Export a Workbook to PDF](#)

 **Important**

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to save a workbook in PDF format using the [Workbook.ExportToPdf](#) method.

C#

```
(ExportActions.cs)
using (FileStream pdfFileStream = new FileStream("Documents\\Document_PDF.pdf", FileMode.Create))
{
    workbook.ExportToPdf(pdfFileStream);
}
```


Visual Basic

```
(ExportActions.vb)
Using pdfFileStream As New FileStream("Documents\\Document_PDF.pdf", FileMode.Create)
    workbook.ExportToPdf(pdfFileStream)
End Using
```


See Also
[Supported Formats](#)

How to: Export a Document to HTML

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Files](#) > [How to: Export a Document to HTML](#)

 **Important**

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

The example below demonstrates how to save a worksheet as HTML file using the [Workbook.ExportToHtml](#) method. To specify export options, create an instance of the `HtmlDocumentExporterOptions` class and pass it as a parameter to the **ExportToHtml** method.

C#

```
(ExportActions.cs)
Worksheet worksheet = workbook.Worksheets["Grouping"];
workbook.Worksheets.ActiveWorksheet = worksheet;
HtmlDocumentExporterOptions options = new HtmlDocumentExporterOptions();
// Specify the part of the document to be exported to HTML.
options.SheetIndex = worksheet.Index;
options.Range = "B2:G7";
// Export the active worksheet to a stream as HTML with the specified options.
using (FileStream htmlStream = new FileStream("OutputWorksheet.html", FileMode.Create))
{
    workbook.ExportToHtml(htmlStream, options);
}
System.Diagnostics.Process.Start("OutputWorksheet.html");
```

Visual Basic

```
(ExportActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Grouping")
workbook.Worksheets.ActiveWorksheet = worksheet
Dim options As New HtmlDocumentExporterOptions()
' Specify the part of the document to be exported to HTML.
options.SheetIndex = worksheet.Index
options.Range = "B2:G7"
' Export the active worksheet to a stream as HTML with the specified options.
Using htmlStream As New FileStream("OutputWorksheet.html", FileMode.Create)
    workbook.ExportToHtml(htmlStream, options)
End Using
System.Diagnostics.Process.Start("OutputWorksheet.html")
```

See Also
[Supported Formats](#)

Workbooks

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Workbooks](#)


This section contains the following example:

- [How to: Create a New Workbook](#)
- [How to: Clone a Workbook](#)
- [How to: Specify Document Properties](#)
- [How to: Merge Multiple Workbooks Into One Document](#)

How to: Create a New Workbook

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Workbooks](#) > [How to: Create a New Workbook](#)

To work with a spreadsheet document programmatically, create an instance of the [Workbook](#) class. This is a root object that serves as the starting point for using a non-visual spreadsheet engine.

 **Important**

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

By default, a new workbook contains one empty worksheet ("Sheet1"). You can [add](#) and [delete](#) worksheets, if required. The same workbook is also created when you call the [Workbook.CreateNewDocument](#) method.


To load an existing spreadsheet document to a workbook, call the [Workbook.LoadDocument](#) method. To save a workbook to a file, use the [Workbook.SaveDocument](#) method. See the [How to: Load a Document to a Workbook](#) and [How to: Save a Document to a File](#) examples.

C#

```
// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
// ...
// Create a new Workbook object.
Workbook workbook = new Workbook();
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
' ...
' Create a new Workbook object.
Private workbook As New Workbook()
```

 **Resolving Performance Issues**

To improve performance while applying multiple modifications to a document, wrap your code in the [Workbook.BeginUpdate-Workbook.EndUpdate](#) method calls.

When you finish working with the [Workbook](#), you are advised to call the [Workbook.Dispose](#) method to release all the resources used by the object. This will allow you to avoid memory leaks and speed up system performance. You can also operate with the [Workbook](#) instance within the **using** statement (**Using** block in Visual Basic).

How to: Clone a Workbook

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Workbooks](#) > [How to: Clone a Workbook](#)

Important

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

The following example demonstrates how to use the [WorkbookExtensions.Clone](#) method to create a workbook copy.

Use the [WorkbookExtensions.Clone](#) method overload with the *copyFormulas* parameter set to **false** to create a copy that replaces formulas with their calculated results.

C#

```
// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
// ...
// Create a new Workbook object.
Workbook workbook = new Workbook();
workbook.LoadDocument("Document.xlsx", DocumentFormat.Xlsx);
// Create a copy of the Workbook object.
Workbook copy = workbook.Clone();
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
' ...
' Create a new Workbook object.
Private workbook As New Workbook()
workbook.LoadDocument("Document.xlsx", DocumentFormat.Xlsx)
' Create a copy of the Workbook object.
Dim copy As Workbook = workbook.Clone()
```

How to: Specify Document Properties

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Workbooks](#) > [How to: Specify Document Properties](#)

Important

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

The document properties are metadata associated and stored with a workbook. To specify the [standard document properties](#) (such as DocumentProperties.Title, DocumentProperties.Author, DocumentProperties.Subject, DocumentProperties.Description etc.), use the [Workbook.DocumentProperties](#) property, which provides access to the DocumentProperties object, containing basic information about a workbook. Note that some of these properties are updated automatically when a document is created (DocumentProperties.Author, DocumentProperties.Created), last modified and saved (DocumentProperties.LastModifiedBy, DocumentProperties.Modified), or printed (DocumentProperties.Printed).

You can also create your own [custom document properties](#) by using the DocumentProperties.Custom property.

Built-In Properties

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to specify the standard document properties for a workbook using the [Workbook.DocumentProperties](#) property.

C#

```
(DocumentPropertiesActions.cs)
// Set the built-in document properties.
workbook.DocumentProperties.Title = "Spreadsheet API: document properties example";
workbook.DocumentProperties.Description = "How to manage document properties using the Spreadsheet API";
workbook.DocumentProperties.Keywords = "Spreadsheet, API, properties, OLEProps";
workbook.DocumentProperties.Company = "Developer Express Inc.";
// Display the specified built-in properties in a worksheet.
worksheet["B3"].Value = "Title";
worksheet["C3"].Value = workbook.DocumentProperties.Title;
worksheet["B4"].Value = "Description";
worksheet["C4"].Value = workbook.DocumentProperties.Description;
worksheet["B5"].Value = "Keywords";
worksheet["C5"].Value = workbook.DocumentProperties.Keywords;
worksheet["B6"].Value = "Company";
worksheet["C6"].Value = workbook.DocumentProperties.Company;
```

Visual Basic

```
(DocumentPropertiesActions.vb)
' Set the built-in document properties.
workbook.DocumentProperties.Title = "Spreadsheet API: document properties example"
workbook.DocumentProperties.Description = "How to manage document properties using the Spreadsheet API"
workbook.DocumentProperties.Keywords = "Spreadsheet, API, properties, OLEProps"
workbook.DocumentProperties.Company = "Developer Express Inc."
' Display the specified built-in properties in a worksheet.
worksheet("B3").Value = "Title"
worksheet("C3").Value = workbook.DocumentProperties.Title
worksheet("B4").Value = "Description"
worksheet("C4").Value = workbook.DocumentProperties.Description
worksheet("B5").Value = "Keywords"
worksheet("C5").Value = workbook.DocumentProperties.Keywords
worksheet("B6").Value = "Company"
worksheet("C6").Value = workbook.DocumentProperties.Company
```

Custom Properties

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>

C#
<pre>(DocumentPropertiesActions.cs) // Set the custom document properties. workbook.DocumentProperties.Custom["Revision"] = 3; workbook.DocumentProperties.Custom["Completed"] = true; workbook.DocumentProperties.Custom["Published"] = DateTime.Now; // Display the specified custom properties in a worksheet. IEnumerable<string> customPropertiesNames = workbook.DocumentProperties.Cu int rowIndex = 2; foreach (string propertyName in customPropertiesNames) { worksheet[rowIndex, 1].Value = propertyName; worksheet[rowIndex, 2].Value = workbook.DocumentProperties.Custom[prop if (worksheet[rowIndex, 2].Value.IsDateTime) worksheet[rowIndex, 2].NumberFormat = "[\$-409]m/d/yyyy h:mm AM/PM" rowIndex++; }</pre>

Visual Basic
<pre>(DocumentPropertiesActions.vb) ' Set the custom document properties. workbook.DocumentProperties.Custom("Revision") = 3 workbook.DocumentProperties.Custom("Completed") = True workbook.DocumentProperties.Custom("Published") = Date.Now ' Display the specified custom properties in a worksheet. Dim customPropertiesNames As IEnumerable(Of String) = workbook.DocumentPro Dim rowIndex As Integer = 2 For Each propertyName As String In customPropertiesNames worksheet(rowIndex, 1).Value = propertyName worksheet(rowIndex, 2).Value = workbook.DocumentProperties.Custom(prop If worksheet(rowIndex, 2).Value.IsDateTime Then worksheet(rowIndex, 2).NumberFormat = "[\$-409]m/d/yyyy h:mm AM/PM" End If rowIndex += 1 Next propertyName</pre>

Use the DocumentCustomProperties.LinkToContent property to associate the desired custom document property with the cell or cell range content, as shown in the code snippet below.

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=E4339>

.

C#	
<pre>(DocumentPropertiesActions.cs) //Define a name to the cell to be linked to the custom property workbook.DefinedNames.Add("checked_by", "E6"); //Connect the custom property with the named cell workbook.DocumentProperties.Custom.LinkToContent("Checked by", "checked by")</pre>	

Visual Basic	
<pre>(DocumentPropertiesActions.vb) 'Define a name to the cell to be linked to the custom property workbook.DefinedNames.Add("checked_by", "E6") 'Connect the custom property with the named cell workbook.DocumentProperties.Custom.LinkToContent("Checked by", "checked by")</pre>	

To remove all the custom document properties from a workbook, use the DocumentCustomProperties.Clear method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>

.

C#	
<pre>(DocumentPropertiesActions.cs) // Remove all custom document properties. workbook.DocumentProperties.Custom.Clear();</pre>	

Visual Basic	
<pre>(DocumentPropertiesActions.vb) ' Remove all custom document properties. workbook.DocumentProperties.Custom.Clear()</pre>	

How to: Merge Multiple Workbooks Into One Document

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Workbooks](#) > [How to: Merge Multiple Workbooks Into One Document](#)

Important

The [Workbook](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the workbook functionality. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

This example demonstrates how to merge data from multiple workbooks into a single document using the [WorkbookExtensions.Merge](#) extension method.

To specify merge options, create a [WorkbookMergeOptions](#) instance and pass it to the [WorkbookExtensions.Merge](#) method as a parameter. This example uses the default options: it copies all worksheets from the workbook named *Document1* into the workbook *Document2* that calls the [WorkbookExtensions.Merge](#) method. If you wish to combine all workbooks into a new summary workbook, use a [WorkbookMergeOptions](#) instance with the [WorkbookMergeOptions.CreateNewWorkbook](#) property set to **true**.

C#

```
// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
// ...
// Create the first workbook.
Workbook book1 = new Workbook();
book1.LoadDocument("Document1.xlsx", DocumentFormat.Xlsx);
// Create the second workbook.
Workbook book2 = new Workbook();
book2.LoadDocument("Document2.xlsx", DocumentFormat.Xlsx);
// Combine two documents.
book2.Merge(WorkbookMergeOptions.Default, book1);
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
' ...
' Create the first workbook.
Private book1 As New Workbook()
book1.LoadDocument("Document1.xlsx", DocumentFormat.Xlsx)
' Create the second workbook.
Private book2 As New Workbook()
book2.LoadDocument("Document2.xlsx", DocumentFormat.Xlsx)
' Combine two documents.
book2.Merge(WorkbookMergeOptions.Default, book1)
```

Worksheets


[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#)

This section contains the following examples:

- [How to: Access a Worksheet](#)
- [How to: Set an Active Worksheet](#)
- [How to: Add a New Worksheet](#)
- [How to: Delete a Worksheet](#)
- [How to: Rename a Worksheet](#)
- [How to: Copy Worksheets within a Workbook](#)
- [How to: Copy Worksheets between Workbooks](#)
- [How to: Move a Worksheet to another Location](#)
- [How to: Show and Hide a Worksheet](#)
- [How to: Show and Hide Gridlines](#)
- [How to: Show and Hide Row and Column Headings](#)
- [How to: Set Page Orientation](#)
- [How to: Set Page Margins](#)
- [How to: Set Paper Size](#)
- [How to: Zoom In and Out of a Worksheet](#)

How to: Access a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Access a Worksheet](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

This example demonstrates how to access worksheets in a workbook. Use the [Workbook.Worksheets](#) property to get a collection of worksheets contained in a workbook (the WorksheetCollection object). To get an individual worksheet by its index or name, use the WorksheetCollection.Item property.

C#

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
// Access a collection of worksheets.
WorksheetCollection worksheets = workbook.Worksheets;
// Access a worksheet by its index.
Worksheet worksheet1 = workbook.Worksheets[0];
// Access a worksheet by its name.
Worksheet worksheet2 = workbook.Worksheets["Sheet2"];
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
' Access a collection of worksheets.
Dim worksheets As WorksheetCollection = workbook.Worksheets
' Access a worksheet by its index.
Dim worksheet1 As Worksheet = workbook.Worksheets(0)
' Access a worksheet by its name.
Dim worksheet2 As Worksheet = workbook.Worksheets("Sheet2")
```

A worksheet index is zero-based. It specifies the worksheet position within a collection. The worksheet name is unique within the collection, and is shown on a worksheet tab. The image below shows worksheet indexes and names in a workbook that is opened in Microsoft® Excel®.




See Also
[How to: Move a Worksheet to another Location](#)
[How to: Rename a Worksheet](#)

How to: Set an Active Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Set an Active Worksheet](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

To set an active worksheet within a workbook, assign the corresponding Worksheet object to the WorksheetCollection.ActiveWorksheet property.

C#


```
(WorksheetActions.cs)
// Set the second worksheet under the "Sheet2" name as active.
workbook.Worksheets.ActiveWorksheet = workbook.Worksheets["Sheet2"];
```

Visual Basic


```
(WorksheetActions.vb)
' Set the second worksheet under the "Sheet2" name as active.
workbook.Worksheets.ActiveWorksheet = workbook.Worksheets("Sheet2")
```

How to: Add a New Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Add a New Worksheet](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to add a new worksheet to a workbook. To do this, use the WorksheetCollection.Add method of the WorksheetCollection collection accessed via [Workbook.Worksheets](#).

To insert a worksheet at the specified position in the WorksheetCollection collection, call the WorksheetCollection.Insert method with the passed worksheet zero-based index.

To specify a worksheet name, use the Worksheet.Name property or pass the worksheet name to the WorksheetCollection.Add or WorksheetCollection.Insert method as a parameter. When naming a worksheet, take into account the constraints listed in the [How to: Rename a Worksheet](#) document.

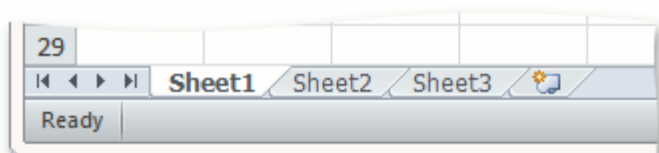
C#

```
(WorksheetActions.cs)
// Add a new worksheet to the workbook. The worksheet will be inserted into the end of the existing worksheets
// under the name "SheetN", where N is a number following the largest number used in worksheet names in the workbook.
workbook.Worksheets.Add();
// Add a new worksheet under the specified name.
workbook.Worksheets.Add().Name = "TestSheet1";
workbook.Worksheets.Add("TestSheet2");
// Add a new worksheet to the specified position in the collection of worksheets.
workbook.Worksheets.Insert(1, "TestSheet3");
workbook.Worksheets.Insert(3);
```

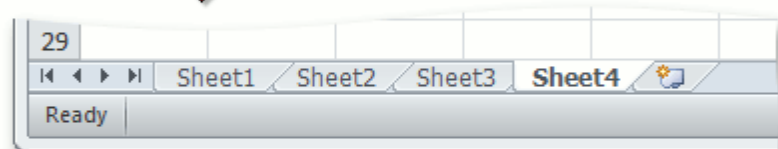
Visual Basic

```
(WorksheetActions.vb)
' Add a new worksheet to the workbook. The worksheet will be inserted into the end of the existing worksheets
' under the name "SheetN", where N is a number following the largest number used in worksheet names in the workbook.
workbook.Worksheets.Add()
' Add a new worksheet under the specified name.
workbook.Worksheets.Add().Name = "TestSheet1"
workbook.Worksheets.Add("TestSheet2")
' Add a new worksheet to the specified position in the collection of worksheets.
workbook.Worksheets.Insert(1, "TestSheet3")
workbook.Worksheets.Insert(3)
```

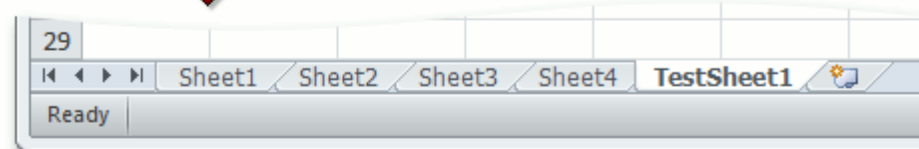
The image below shows the result (the modified file is opened in Microsoft® Excel®).



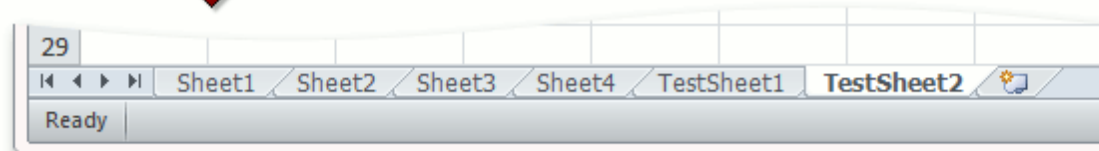
```
workbook.Worksheets.Add();
```



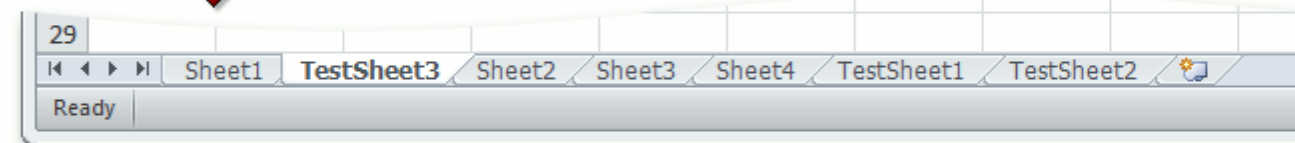
```
workbook.Worksheets.Add().Name = "TestSheet1";
```



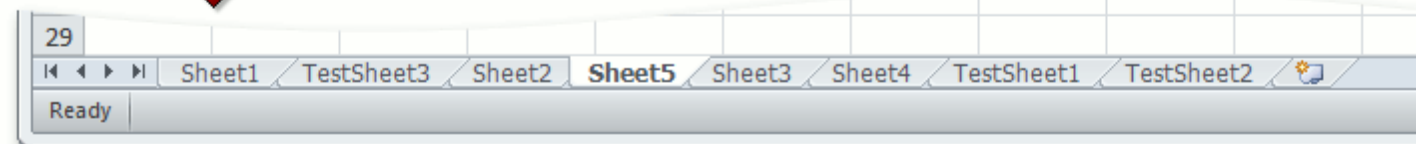
```
workbook.Worksheets.Add("TestSheet2");
```



```
workbook.Worksheets.Insert(1, "TestSheet3");
```



```
workbook.Worksheets.Insert(3);
```




See Also

[How to: Rename a Worksheet](#)


[How to: Delete a Worksheet](#)

How to: Delete a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Delete a Worksheet](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to remove a worksheet from a workbook by calling the `WorksheetCollection.Remove` or `WorksheetCollection.RemoveAt` method of the `WorksheetCollection` object that is accessed via the [Workbook.Worksheets](#) property.


C#

```
(WorksheetActions.cs)
// Delete the "Sheet2" worksheet from the workbook.
workbook.Worksheets.Remove(workbook.Worksheets["Sheet2"]);
// Delete the first worksheet from the workbook.
workbook.Worksheets.RemoveAt(0);
```

Visual Basic

```
(WorksheetActions.vb)
' Delete the "Sheet2" worksheet from the workbook.
workbook.Worksheets.Remove(workbook.Worksheets("Sheet2"))
' Delete the first worksheet from the workbook.
workbook.Worksheets.RemoveAt(0)
```

You can also hide a worksheet. See the [How to: Show and Hide a Worksheet](#) topic for details.


 **Note**

A workbook must always contain at least one visible worksheet.


See Also
[How to: Show and Hide a Worksheet](#)
[How to: Add a New Worksheet](#)

How to: Rename a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Rename a Worksheet](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to rename a worksheet. To do this, use the Worksheet.Name property of the corresponding Worksheet object.

C#

(WorksheetActions.cs)
// Change the name of the second worksheet.
workbook.Worksheets[1].Name = "Renamed Sheet";


Visual Basic

(WorksheetActions.vb)
' Change the name of the second worksheet.
workbook.Worksheets(1).Name = "Renamed Sheet"


- When naming a worksheet, take into account the following constraints.
- The worksheet name must not be equal to an existing name in the collection of worksheets ([Workbook.Worksheets](#)).
 - The worksheet name must not exceed 31 characters.
 - The worksheet name must not contain the following symbols: \, /, ?, :, *, [or]
 - The worksheet name must not start and end with a single quote (').
 - The worksheet name must not be an empty string.

How to: Copy Worksheets within a Workbook

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Copy Worksheets within a Workbook](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to create a copy of an existing worksheet. To do this, follow the steps below.

- [Create an empty worksheet in a workbook](#). For example, call the WorksheetCollection.Add method.
- Access the newly created worksheet object and call its Worksheet.CopyFrom method with the source worksheet object passed as a parameter.

C#

(WorksheetActions.cs)
// Add a new worksheet to a workbook.
workbook.Worksheets.Add("Sheet1_Copy");
// Copy all information (content and formatting) to the newly created worksheet
// from the "Sheet1" worksheet.
workbook.Worksheets["Sheet1_Copy"].CopyFrom(workbook.Worksheets["Sheet1"]);


Visual Basic

(WorksheetActions.vb)
' Add a new worksheet to a workbook.
workbook.Worksheets.Add("Sheet1_Copy")
' Copy all information (content and formatting) to the newly created works
' from the "Sheet1" worksheet.
workbook.Worksheets("Sheet1_Copy").CopyFrom(workbook.Worksheets("Sheet1"))


See Also
[How to: Copy Worksheets between Workbooks](#)

How to: Copy Worksheets between Workbooks

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Copy Worksheets between Workbooks](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to add an existing worksheet to a new workbook. To do this, access a Worksheet object that specifies a worksheet into which you want to copy another worksheet and call its Worksheet.CopyFrom method with the source worksheet object passed as a parameter.

C#

```
(WorksheetActions.cs)
// Create a source workbook.
Workbook sourceWorkbook = new Workbook();
// Add a new worksheet.
sourceWorkbook.Worksheets.Add();
// Modify the second worksheet of the source workbook to be copied.
sourceWorkbook.Worksheets[1].Cells["A1"].Value = "A worksheet to be copied";
sourceWorkbook.Worksheets[1].Cells["A1"].Font.Color = Color.ForestGreen;
// Copy the second worksheet of the source workbook into the first worksheet of another workbook.
workbook.Worksheets[0].CopyFrom(sourceWorkbook.Worksheets[1]);
```


Visual Basic

```
(WorksheetActions.vb)
' Create a source workbook.
Dim sourceWorkbook As New Workbook()
' Add a new worksheet.
sourceWorkbook.Worksheets.Add()
' Modify the second worksheet of the source workbook to be copied.
sourceWorkbook.Worksheets(1).Cells("A1").Value = "A worksheet to be copied"
sourceWorkbook.Worksheets(1).Cells("A1").Font.Color = Color.ForestGreen
' Copy the second worksheet of the source workbook into the first worksheet of another workbook.
workbook.Worksheets(0).CopyFrom(sourceWorkbook.Worksheets(1))
```


See Also
[How to: Copy Worksheets within a Workbook](#)

How to: Move a Worksheet to another Location

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Move a Worksheet to another Location](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to move a worksheet to another location in a workbook. To do this, call the `Worksheet.Move` method and pass the worksheet zero-based index specifying the position in the `WorksheetCollection` collection into which the worksheet should be moved.

C#

```
(WorksheetActions.cs)
// Move the first worksheet to the position of the last worksheet within the workbook.
workbook.Worksheets[0].Move(workbook.Worksheets.Count - 1);
```

Visual Basic

```
(WorksheetActions.vb)
' Move the first worksheet to the position of the last worksheet within the workbook.
workbook.Worksheets(0).Move(workbook.Worksheets.Count - 1)
```

You can also rearrange worksheets within a workbook by using methods inherited from the `Sheet` interface: `Sheet.MoveToBeginning`, `Sheet.MoveBefore`, `Sheet.MoveAfter`, and `Sheet.MoveToEnd`.

C#

```
workbook.Worksheets[0].MoveToEnd();
```

Visual Basic

```
workbook.Worksheets(0).MoveToEnd()
```

The image below shows how the worksheet moves from the first position to the last position in the workbook (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					

◀ ▶

Sheet1

Sheet2

Sheet3

+

0 1 2 ← Worksheet Indexes



	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					

◀ ▶

Sheet2

Sheet3

Sheet1

+

0 1 2 ← Worksheet Indexes

How to: Show and Hide a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Show and Hide a Worksheet](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to manage worksheet visibility in a workbook. To do this, use the following properties.

- Worksheet.Visible

Set this property to **false** to hide a worksheet. To make a worksheet visible again, you can set the **Visible** property to **true** or the Worksheet.VisibilityType property to the WorksheetVisibilityType.Visible enumeration member. End-users can show a hidden worksheet via the user interface (for example, when opening a workbook in Microsoft® Excel® or SpreadsheetControl).

- Worksheet.VisibilityType

Setting this property to WorksheetVisibilityType.Hidden is equivalent to setting the Worksheet.Visible property to **false** - the hidden worksheet can be shown again by end-users via the user interface (for example, Microsoft® Excel® or SpreadsheetControl).

The **VisibilityType** property also allows you to hide a worksheet so that it becomes impossible for end-users to access this worksheet. To do this, mark a worksheet as "very hidden" by setting the **VisibilityType** property to WorksheetVisibilityType.VeryHidden. To show a worksheet again, set the **Visible** property to **true** or **VisibilityType** to WorksheetVisibilityType.Visible.

Note

A workbook must always contain at least one visible worksheet.

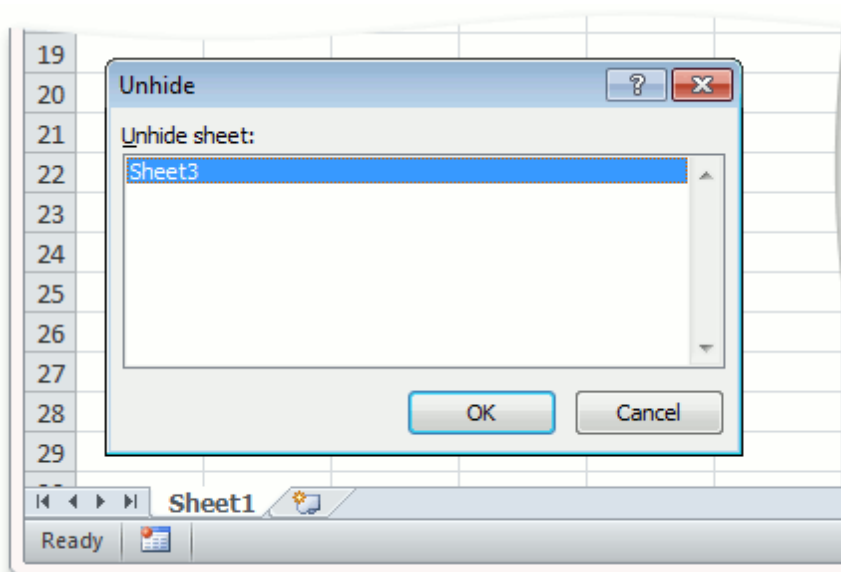
C#

(WorksheetActions.cs)
// Hide the worksheet under the "Sheet2" name and prevent end-users from unhiding it via user interface.
// To make this worksheet visible again, use the Worksheet.Visible property.
workbook.Worksheets["Sheet2"].VisibilityType = WorksheetVisibilityType.VeryHidden;
// Hide the "Sheet3" worksheet.
// In this state a worksheet can be unhiden via user interface.
workbook.Worksheets["Sheet3"].Visible = false;

Visual Basic

(WorksheetActions.vb)
' Hide the worksheet under the "Sheet2" name and prevent end-users from un
' To make this worksheet visible again, use the Worksheet.Visible property
workbook.Worksheets("Sheet2").VisibilityType = WorksheetVisibilityType.Ver
' Hide the "Sheet3" worksheet.
' In this state a worksheet can be unhiden via user interface.
workbook.Worksheets("Sheet3").Visible = False

After executing the code above, the "Sheet2" worksheet is hidden, and cannot be accessed by an end-user in Microsoft® Excel®. The "Sheet3" worksheet is also hidden, but its visibility can be restored by an end-user.

**See Also**


[How to: Delete a Worksheet](#)

How to: Show and Hide Gridlines

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Show and Hide Gridlines](#)


 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to control the visibility of the worksheet gridlines. To do this, use the `WorksheetView.ShowGridlines` property of the worksheet view object that is accessed via the `Worksheet.ActiveView` property.

 **Note**

To specify whether the worksheet gridlines should be printed, use the `WorksheetPrintOptions.PrintGridlines` property of the object accessed via `Worksheet.PrintOptions`. Refer to the [How to: Specify Print Settings](#) document for details on how to specify print options for a worksheet.

C#

```
(WorksheetActions.cs)
// Hide gridlines on the first worksheet.
workbook.Worksheets[0].ActiveView.ShowGridlines = false;
```

Visual Basic

```
(WorksheetActions.vb)
' Hide gridlines on the first worksheet.
workbook.Worksheets(0).ActiveView.ShowGridlines = False
```

The image below shows the worksheet when gridlines are displayed, and when gridlines are hidden (the workbook is opened in Microsoft® Excel®).

Worksheet.ActiveView.ShowGridlines = true

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Worksheet.ActiveView.ShowGridlines = false


	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

How to: Show and Hide Row and Column Headings

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Show and Hide Row and Column Headings](#)


 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

Each worksheet (Worksheet) consists of cells (Worksheet.Cells) that are arranged in rows (Worksheet.Rows) and columns (Worksheet.Columns). Each row and column has its own unique name (numbers "1", "2", "3", etc. are used for rows and letters A", "B", "C", etc. are used for columns). These unique names for rows and columns are displayed as headings at the left and at the top of a worksheet, respectively. To control the visibility of row and column headings on a worksheet, use the `WorksheetView.ShowHeadings` property of the worksheet view object that is accessed via the `Worksheet.ActiveView` property.

 **Note**

To specify whether row and column headings should be printed, use the `WorksheetPrintOptions.PrintHeadings` property of the object accessed via `Worksheet.PrintOptions`. Refer to the [How to: Specify Print Settings](#) document for details on how to specify print options for a worksheet.

C#

```
(WorksheetActions.cs)
// Hide row and column headings in the first worksheet.
workbook.Worksheets[0].ActiveView.ShowHeadings = false;
```

Visual Basic

```
(WorksheetActions.vb)
' Hide row and column headings in the first worksheet.
workbook.Worksheets(0).ActiveView.ShowHeadings = False
```

The image below shows the appearance of the worksheet when row and column headings are shown, and when row and column headings are hidden (the workbook is opened in Microsoft® Excel®).


Worksheet.ActiveView.ShowHeadings = true

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					


Worksheet.ActiveView.ShowHeadings = false

How to: Set Page Orientation

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Set Page Orientation](#)


 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to set the orientation of worksheet pages via the `WorksheetView.Orientation` property of the worksheet view object that is accessed via the `Worksheet.ActiveView`. The `PageOrientation` enumerator lists available page orientation types.

 **Note**

The `WorksheetView.Orientation` property value is used when worksheet pages are printed. To learn how to specify other worksheet print options, see the [How to: Specify Print Settings](#) document.

C#

```
(WorksheetActions.cs)
// Set the page orientation to Landscape.
workbook.Worksheets[0].ActiveView.Orientation = PageOrientation.Landscape;
```


Visual Basic

```
(WorksheetActions.vb)
' Set the page orientation to Landscape.
workbook.Worksheets(0).ActiveView.Orientation = PageOrientation.Landscape
```


See Also
[How to: Set Page Margins](#)
[How to: Set Paper Size](#)

How to: Set Page Margins

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Set Page Margins](#)


 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to set margins of worksheet pages via properties of the Margins object. To access this object, use the WorksheetView.Margins property of the worksheet view object that is returned by the Worksheet.ActiveView. To select units of measure for worksheet page margins, set the [Workbook.Unit](#) property to the required DocumentUnit enumeration member.

 **Note**

The WorksheetView.Margins settings are used when worksheet pages are printed. To learn how to specify other worksheet print options, see the [How to: Specify Print Settings](#) document.

C#

```
(WorksheetActions.cs)
// Select a unit of measure used within the workbook.
workbook.Unit = DevExpress.Office.DocumentUnit.Inch;
// Access page margins.
Margins pageMargins = workbook.Worksheets[0].ActiveView.Margins;
// Specify page margins.
pageMargins.Left = 1;
pageMargins.Top = 1.5F;
pageMargins.Right = 1;
pageMargins.Bottom = 0.8F;
// Specify header and footer margins.
pageMargins.Header = 1;
pageMargins.Footer = 0.4F;
```


Visual Basic

```
(WorksheetActions.vb)
' Select a unit of measure used within the workbook.
workbook.Unit = DevExpress.Office.DocumentUnit.Inch
' Access page margins.
Dim pageMargins As Margins = workbook.Worksheets(0).ActiveView.Margins
' Specify page margins.
pageMargins.Left = 1
pageMargins.Top = 1.5F
pageMargins.Right = 1
pageMargins.Bottom = 0.8F
' Specify header and footer margins.
pageMargins.Header = 1
pageMargins.Footer = 0.4F
```


See Also
[How to: Set Page Orientation](#)
[How to: Set Paper Size](#)

How to: Set Paper Size

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Set Paper Size](#)


 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to set the paper size of worksheet pages via the `WorksheetView.PaperKind` property of the worksheet view object accessed via `Worksheet.ActiveView`. The [PaperKind](#) enumerator lists available page sizes.

 **Note**

The `WorksheetView.PaperKind` property value is used when worksheet pages are printed. To learn how to specify other worksheet print options, see the [How to: Specify Print Settings](#) document.

C#

```
(WorksheetActions.cs)
// Select the page's paper size.
workbook.Worksheets[0].ActiveView.PaperKind = System.Drawing.Printing.PaperKind.A4;
```


Visual Basic

```
(WorksheetActions.vb)
' Select the page's paper size.
workbook.Worksheets(0).ActiveView.PaperKind = System.Drawing.Printing.Pape
```


See Also
[How to: Set Page Orientation](#)
[How to: Set Page Margins](#)

How to: Zoom In and Out of a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Worksheets](#) > [How to: Zoom In and Out of a Worksheet](#)


 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to zoom a worksheet view (Worksheet.ActiveView). To do this, set the WorksheetView.Zoom property to an integer value that specifies the required zoom factor for a worksheet view.

 **Note**

To specify how a worksheet should be scaled when it is printed, use the WorksheetPrintOptions.Scale property of the object accessed via Worksheet.PrintOptions. Refer to the [How to: Specify Print Settings](#) document for details on how to specify print options for a worksheet.

C#

```
(WorksheetActions.cs)
// Zoom in the worksheet view.
workbook.Worksheets[0].ActiveView.Zoom = 150;
```

Visual Basic

```
(WorksheetActions.vb)
' Zoom in the worksheet view.
workbook.Worksheets(0).ActiveView.Zoom = 150
```

The image below shows the result (the workbook is opened in Microsoft® Excel®).



Rows and Columns

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Rows and Columns](#)

This section contains the following examples:

- [How to: Access a Row or Column](#)
- [How to: Add a New Row or Column to a Worksheet](#)
- [How to: Delete a Row or Column from a Worksheet](#)
- [How to: Copy a Row or Column](#)
- [How to: Hide a Row or a Column](#)
- [How to: Specify Row Height or Column Width](#)

How to: Access a Row or Column

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Rows and Columns](#) > [How to: Access a Row or Column](#)

Rows

This example demonstrates how to access rows in a worksheet. Use the Worksheet.Rows property to get a collection of rows contained in a worksheet (the RowCollection object). To get an individual row by its index (zero-based) or heading ("1", "2", "3", etc.), use the RowCollection.Item property.

C#

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
// Access a collection of rows.
RowCollection rows = workbook.Worksheets[0].Rows;
// Access the first row by its index in the collection of rows.
Row firstRow_byIndex = rows[0];
// Access the first row by its unique name.
Row firstRow_byName = rows["1"];
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
' Access a collection of rows.
Dim rows As RowCollection = workbook.Worksheets(0).Rows
' Access the first row by its index in the collection of rows.
Dim firstRow_byIndex As Row = rows(0)
' Access the first row by its unique name.
Dim firstRow_byName As Row = rows("1")
```

Columns

This example demonstrates how to access columns in a worksheet. Use the Worksheet.Columns property to get a collection of columns contained in a worksheet (the ColumnCollection object). To get an individual column by its index (zero-based) or heading ("A", "B", "C", etc.), use the ColumnCollection.Item property.

C#

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
// Access a collection of columns.
ColumnCollection columns = workbook.Worksheets[0].Columns;
// Access the first column by its index in the collection of columns.
Column firstColumn_byIndex = columns[0];
// Access the first column by its unique name.
Column firstColumn_byName = columns["A"];
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
' Access a collection of columns.
Dim columns As ColumnCollection = workbook.Worksheets(0).Columns
' Access the first column by its index in the collection of columns.
Dim firstColumn_byIndex As Column = columns(0)
' Access the first column by its unique name.
Dim firstColumn_byName As Column = columns("A")
```

The image below shows row and column indexes and headings in a worksheet, when a workbook is opened in Microsoft® Excel®.

Row
Indexes

↓

0

1

2

3

4

5

6

7

...

	0	1	2	3	4	5	6...
	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							
7							
8							

↑

Row
Headings

← Column Indexes

← Column Headings

See Also
[Rows and Columns](#)

How to: Add a New Row or Column to a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Rows and Columns](#) > [How to: Add a New Row or Column to a Worksheet](#)

Row

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to insert new rows into a worksheet. You can do this in the following ways.

- Use the Row.Insert method.
- Call the RowCollection.Insert method of the object accessed via the Worksheet.Rows property. To specify where a new row should be inserted, pass a zero-based row index to the method as a parameter. To insert multiple rows, pass the number of rows to be inserted as well.
- To insert empty rows above the specified range of cells, use the Worksheet.InsertCells method with the cell range and InsertCellsMode.EntireRow enumeration member passed as parameters. This method inserts the same number of rows that contains the specified cell range.

C#

```
(RowAndColumnActions.cs)
// Insert a new row 3.
worksheet.Rows["3"].Insert();
// Insert a new row into the worksheet at the 5th position.
worksheet.Rows.Insert(4);
// Insert five rows into the worksheet at the 9th position.
worksheet.Rows.Insert(8, 5);
// Insert two rows above the "L15:M16" cell range.
worksheet.InsertCells(worksheet.Range["L15:M16"], InsertCellsMode.EntireRow);
```

Visual Basic

```
(RowAndColumnActions.vb)
' Insert a new row 3.
worksheet.Rows("3").Insert()
' Insert a new row into the worksheet at the 5th position.
worksheet.Rows.Insert(4)
' Insert five rows into the worksheet at the 9th position.
worksheet.Rows.Insert(8, 5)
' Insert two rows above the "L15:M16" cell range.
worksheet.InsertCells(worksheet.Range("L15:M16"), InsertCellsMode.EntireRow)
```

Note

The number of rows in a worksheet is permanently fixed - 1048576.

Column

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>

This example demonstrates how to insert new columns into a worksheet. You can do this in the following ways.

- Use the Column.Insert method.

- Call the ColumnCollection.Insert method of the object accessed via the Worksheet.Columns property. To specify where a new column should be inserted, pass a zero-based column index to the method as a parameter. To insert multiple columns, pass the number of columns to be inserted as well.
- To insert empty columns to the left of the specified range of cells, use the Worksheet.InsertCells method with the cell range and InsertCellsMode.EntireColumn enumeration member passed as parameters. This method inserts the same number of columns that comprising the specified cell range.

C#	
<pre>(RowAndColumnActions.cs) // Insert a new column C. worksheet.Columns["C"].Insert(); // Insert a new column into the worksheet at the 5th position. worksheet.Columns.Insert(4); // Insert three columns into the worksheet at the 7th position. worksheet.Columns.Insert(6, 3); // Insert two columns to the left of the "L15:M16" cell range. worksheet.InsertCells(worksheet.Range["L15:M16"], InsertCellsMode.EntireCo</pre>	

Visual Basic	
<pre>(RowAndColumnActions.vb) ' Insert a new column C. worksheet.Columns("C").Insert() ' Insert a new column into the worksheet at the 5th position. worksheet.Columns.Insert(4) ' Insert three columns into the worksheet at the 7th position. worksheet.Columns.Insert(6, 3) ' Insert two columns to the left of the "L15:M16" cell range. worksheet.InsertCells(worksheet.Range("L15:M16"), InsertCellsMode.EntireCo</pre>	

Note	
The number of columns in a worksheet is permanently fixed - 16384.	

See Also
Rows and Columns
[How to: Delete a Row or Column from a Worksheet](#)

How to: Delete a Row or Column from a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Rows and Columns](#) > [How to: Delete a Row or Column from a Worksheet](#)

Row

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to delete rows from a worksheet. You can do this in one of the following ways.

- Call the Row.Delete method of the Row object representing the row to be deleted.
- Call the RowCollection.Remove method of the object that is accessed via the Worksheet.Rows property. Pass the zero-based index of a row to be deleted as a parameter. To delete multiple rows, pass the number of rows to be deleted as well.
- To delete a row containing the specified cell or multiple rows containing the specified range of cells, use the Worksheet.DeleteCells method with the cell or cell range and DeleteMode.EntireRow enumeration member passed as parameters.

When you delete rows from a worksheet, other rows are automatically shifted up.

C#

(RowAndColumnActions.cs)
// Delete the 2nd row from the worksheet.
worksheet.Rows[1].Delete();
// Delete the 3rd row from the worksheet.
worksheet.Rows.Remove(2);
// Delete three rows from the worksheet starting from the 10th row.
worksheet.Rows.Remove(9, 3);
// Delete a row that contains the "B2" cell.
worksheet.DeleteCells(worksheet.Cells["B2"], DeleteMode.EntireRow);

Visual Basic

(RowAndColumnActions.vb)
' Delete the 2nd row from the worksheet.
worksheet.Rows(1).Delete()
' Delete the 3rd row from the worksheet.
worksheet.Rows.Remove(2)
' Delete three rows from the worksheet starting from the 10th row.
worksheet.Rows.Remove(9, 3)
' Delete a row that contains the "B2" cell.
worksheet.DeleteCells(worksheet.Cells("B2"), DeleteMode.EntireRow)

You can also hide worksheet rows. See the [How to: Hide a Row or a Column](#) topic for details.

Note

The number of rows in a worksheet is permanently fixed - 1048576.

Column

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>

.

This example demonstrates how to delete columns from a worksheet. You can do this in one of the following ways.

- Call the Column.Delete method of the Column object representing the column to be deleted.
- Call the ColumnCollection.Remove method of the object that is accessed via the Worksheet.Columns property. Pass the zero-based index of a column to be deleted as a parameter. To delete multiple columns, pass the number of columns to be deleted as well.
- To delete a column containing the specified cell or multiple columns containing the specified range of cells, use the Worksheet.DeleteCells method with the cell or cell range and DeleteMode.EntireColumn enumeration member passed as parameters.

When you delete columns from a worksheet, other columns are automatically shifted to the left.

C#	
<pre>(RowAndColumnActions.cs) // Delete the 2nd column from the worksheet. worksheet.Columns[1].Delete(); // Delete the 3rd column from the worksheet. worksheet.Columns.Remove(2); // Delete three columns from the worksheet starting from the 10th column. worksheet.Columns.Remove(9, 3); // Delete a column that contains the "B2"cell. worksheet.DeleteCells(worksheet.Cells["B2"], DeleteMode.EntireColumn);</pre>	
Visual Basic	
<pre>(RowAndColumnActions.vb) ' Delete the 2nd column from the worksheet. worksheet.Columns(1).Delete() ' Delete the 3rd column from the worksheet. worksheet.Columns.Remove(2) ' Delete three columns from the worksheet starting from the 10th column. worksheet.Columns.Remove(9, 3) ' Delete a column that contains the "B2"cell. worksheet.DeleteCells(worksheet.Cells("B2"), DeleteMode.EntireColumn)</pre>	

You can also hide worksheet columns. See the [How to: Hide a Row or a Column](#) topic for details.

Note

The number of columns in a worksheet is permanently fixed - 16384.

See Also
Rows and Columns
[How to: Hide a Row or a Column](#)
[How to: Add a New Row or Column to a Worksheet](#)

How to: Copy a Row or Column

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Rows and Columns](#) > [How to: Copy a Row or Column](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to copy a row or column in a worksheet. To do this, access a destination row (Row) or column (Column) object and call its Range.CopyFrom method. Pass the source row or column object as a parameter. If you need to specify the type of data be copied (for example, values only, formats only, formulas only, borders only, etc.), pass the required member of the PasteSpecial enumerator as well.

C#

```
(RowAndColumnActions.cs)
// Copy all data from the 2nd row to the 5th row.
worksheet.Rows["5"].CopyFrom(worksheet.Rows["2"]);
// Copy only borders from the "B" column to the "E" column.
worksheet.Columns["E"].CopyFrom(worksheet.Columns["B"], PasteSpecial.Borders);
```

Visual Basic

```
(RowAndColumnActions.vb)
' Copy all data from the 2nd row to the 5th row.
worksheet.Rows("5").CopyFrom(worksheet.Rows("2"))
' Copy only borders from the "B" column to the "E" column.
worksheet.Columns("E").CopyFrom(worksheet.Columns("B"), PasteSpecial.Borde
```

- See Also**
- [How to: Copy Cell Data Only, Cell Style Only, or Cell Data with Style](#)
 - [How to: Copy Worksheets within a Workbook](#)
 - [How to: Copy Worksheets between Workbooks](#)

How to: Hide a Row or a Column

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Rows and Columns](#) > [How to: Hide a Row or a Column](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to hide rows and columns in a worksheet using different properties and methods:

- Use the Row.Visible and Column.Visible properties
- Call the RowCollection.Hide or the ColumnCollection.Hide methods.
- Set the row height (Row.Height) or column width (Column.Width, Column.WidthInCharacters or Column.WidthInPixels) to **0 (zero)** (see also the [How to: Specify Row Height or Column Width](#) document).

C#

```
(RowAndColumnActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
// Hide the 8th row of the worksheet.
worksheet.Rows[7].Visible = false;
// Hide the 4th column of the worksheet.
worksheet.Columns[3].Visible = false;
// Hide columns from 5 to 7.
worksheet.Columns.Hide(5, 7);
// Hide rows from 6 to 8.
worksheet.Rows.Hide(5, 7);
// Hide the 10th row of the worksheet.
worksheet.Rows[9].Height = 0;
// Hide the 10th column of the worksheet.
worksheet.Columns[9].Width = 0;
```

Visual Basic

```
(RowAndColumnActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Hide the 8th row of the worksheet.
worksheet.Rows(7).Visible = False
' Hide the 4th column of the worksheet.
worksheet.Columns(3).Visible = False
' Hide columns from 5 to 7.
worksheet.Columns.Hide(5, 7)
' Hide rows from 6 to 8.
worksheet.Rows.Hide(5, 7)
' Hide the 10th row of the worksheet.
worksheet.Rows(9).Height = 0
' Hide the 10th column of the worksheet.
worksheet.Columns(9).Width = 0
```

See Also
[How to: Specify Row Height or Column Width](#)

How to: Specify Row Height or Column Width

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Rows and Columns](#) > [How to: Specify Row Height or Column Width](#)

Row

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to control the row height on a worksheet.

- **Set Height for Individual Rows**

Use the row's `Row.Height` property or the `Range.RowHeight` property of the cell or cell range object to set the row height in the units of measure specified by [Workbook.Unit](#).

Note

If the row height is set to 0, the row is hidden. You can also use the `Row.Visible` property to hide a row or display the hidden row again (see [How to: Hide a Row or a Column](#)).

- **AutoFit Row Height**

To automatically change the row height to fit the contents, use the row's `Row.AutoFit` method. To quickly autofit several rows on a worksheet, use the `RowCollection.AutoFit` method.

- **Match the Height of Two Rows**

To match one row height to another, make their `Row.Height` property values equal.

- **Set the Default Row Height**

To set the default height for worksheet rows, use the `Worksheet.DefaultRowHeight` property. This property does not affect rows with an explicitly specified height.

- **Set the Height for All Rows on a Worksheet**

To change the height of all rows on a worksheet, use the `Range.RowHeight` property of the `Column` object corresponding to any column on a worksheet. This sets the height of all rows to the same value, overriding all previously applied row height settings.

C#

```
(RowAndColumnActions.cs)
// Set the height of the 3rd row to 50 points.
workbook.Unit = DevExpress.Office.DocumentUnit.Point;
worksheet.Rows[2].Height = 50;
// Set the height of the row that contains the "C5" cell to 2 inches.
workbook.Unit = DevExpress.Office.DocumentUnit.Inch;
worksheet.Cells["C5"].RowHeight = 2;
// Set the height of the 7th row to the height of the 3rd row.
worksheet.Rows["7"].Height = worksheet.Rows["3"].Height;
// Set the default row height to 30 points.
workbook.Unit = DevExpress.Office.DocumentUnit.Point;
worksheet.DefaultRowHeight = 30;
```

Visual Basic

```
(RowAndColumnActions.vb)
' Set the height of the 3rd row to 50 points.
workbook.Unit = DevExpress.Office.DocumentUnit.Point
worksheet.Rows(2).Height = 50
' Set the height of the row that contains the "C5" cell to 2 inches.
workbook.Unit = DevExpress.Office.DocumentUnit.Inch
worksheet.Cells("C5").RowHeight = 2
' Set the height of the 7th row to the height of the 3rd row.
worksheet.Rows("7").Height = worksheet.Rows("3").Height
' Set the default row height to 30 points.
workbook.Unit = DevExpress.Office.DocumentUnit.Point
worksheet.DefaultRowHeight = 30
```

Column

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>

.

This example demonstrates how to control the column width on a worksheet.

• Set the Width for Individual Columns

To specify a column width in characters of the default font specified by the built-in *Normal* style, use the column's Column.WidthInCharacters property, or the Range.ColumnWidthInCharacters property of the cell or cell range object.

To specify a column width in pixels, use the column's Column.WidthInPixels property.

To set a column width in other measurement units, first set the Workbook.Unit property to the DocumentUnit enumeration member that corresponds to the desired units, then use the column's Column.Width property, or the Range.ColumnWidth property of the cell or cell range object.

Note

If a column width is set to 0, the column is hidden. You can also use the Column.Visible property to hide a column or display the hidden column again (see the [How to: Hide a Row or a Column](#) document).

• AutoFit Column Width

To automatically change the column width to fit the contents, use the columns's Column.AutoFit method. To quickly autofit several columns on a worksheet, use the ColumnCollection.AutoFit method.

• Match the Width of Two Columns

To match one column width to another, you can use the Column.Width property or Range.CopyFrom method of the column object.

• Set the Default Column Width

To set the default width for worksheet columns, use the Worksheet.DefaultColumnWidthInCharacters, Worksheet.DefaultColumnWidthInPixels or Worksheet.DefaultColumnWidth property. These properties do not affect columns with an explicitly specified width.

• Set the Width for All Columns on a Worksheet

To change the width of all columns on a worksheet, use the Range.ColumnWidthInCharacters or Range.ColumnWidth property of the Row object corresponding to any row on a worksheet. This sets the width of all columns to the same value, overriding all previously applied column width settings.

C#	
----	--

```

(RowAndColumnActions.cs)
// Set the "B" column width to 30 characters of the default font that is s
worksheet.Columns["B"].WidthInCharacters = 30;
// Set the "C" column width to 15 millimeters.
worksheet.Columns["C"].Width = 15;
// Set the width of the column that contains the "E15" cell to 100 points.
worksheet.Columns["E15"].ColumnWidth = 100;
// Set the width of all columns that contain the "F4:H7" cell range (the "
worksheet.Range["F4:H7"].ColumnWidth = 70;
// Set the "J" column width to the "B" column width value.
worksheet.Columns["J"].Width = worksheet.Columns["B"].Width;
// Copy the "C" column width value and assign it to the "K" column width.
worksheet.Columns["K"].CopyFrom(worksheet.Columns["C"], PasteSpecial.Column
// Set the default column width to 40 pixels.
worksheet.DefaultColumnWidthInPixels = 40;

```

Visual Basic

```

(RowAndColumnActions.vb)
' Set the "B" column width to 30 characters of the default font that is sp
worksheet.Columns("B").WidthInCharacters = 30
' Set the "C" column width to 15 millimeters.
worksheet.Columns("C").Width = 15
' Set the width of the column that contains the "E15" cell to 100 points.
worksheet.Columns("E15").ColumnWidth = 100
' Set the width of all columns that contain the "F4:H7" cell range (the "F
worksheet.Range("F4:H7").ColumnWidth = 70
' Set the "J" column width to the "B" column width value.
worksheet.Columns("J").Width = worksheet.Columns("B").Width
' Copy the "C" column width value and assign it to the "K" column width.
worksheet.Columns("K").CopyFrom(worksheet.Columns("C"), PasteSpecial.Column
' Set the default column width to 40 pixels.
worksheet.DefaultColumnWidthInPixels = 40

```

Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#)

This section contains the following examples:

- [How to: Access a Cell in a Worksheet](#)
- [How to: Access a Range of Cells](#)
- [How to: Insert a Cell or Cell Range](#)
- [How to: Delete a Cell or Range of Cells](#)
- [How to: Create a Named Range of Cells](#)
- [How to: Change a Cell or Cell Range Value](#)
- [How to: Add Formulas to Cells](#)
- [How to: Add a Hyperlink to a Cell](#)
- [How To: Add a Comment To a Cell](#)
- [How to: Clear Cells of Content, Formatting, Hyperlinks and Comments](#)
- [How to: Copy Cell Data Only, Cell Style Only, or Cell Data with Style](#)
- [How to: Merge Cells or Split Merged Cells](#)

How to: Access a Cell in a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Access a Cell in a Worksheet](#)

This example demonstrates several ways to access individual cells.

- By row and column indexes of a cell via the Worksheet.Item property.
- By a cell reference (for example, A1, B2, etc.) or by row and column indexes from the worksheet's collection of cells. Use the CellCollection.Item property.
- By a column index or column heading from the specified row. Use the Row.Item property.

By a row index or row heading from the specified column. Use the Column.Item property.

See the [How to: Access a Row or Column](#) example to learn how to get an individual row or column.

- By a cell index or by row and column indexes from the specified range of cells. Use the Range.Item property. See the [How to: Access a Range of Cells](#) document to learn how to access an individual range of cells in a worksheet.

C#

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
Worksheet worksheet = workbook.Worksheets[0];
Cell cellA1 = worksheet[0, 0]; // Cell A1
Cell cellB2 = worksheet.Cells["B2"]; // Cell B2
Cell cellC3 = worksheet.Cells[2, 2]; // Cell C3
Cell cellD4 = worksheet.Rows[3][3]; // Cell D4
Cell cellE5 = worksheet.Rows[4]["E"]; // Cell E5
Cell cellF6 = worksheet.Rows["6"][5]; // Cell F6
Cell cellG7 = worksheet.Columns[6][6]; // Cell G7
Cell cellH8 = worksheet.Columns["H"][7]; // Cell H8
Cell cellI9 = worksheet.Columns["I"]["9"]; // Cell I9
// Access a cell from the range of cells.
Cell cellB5 = worksheet.Range["B3:D8"][6]; // Cell B5
Cell cellD6 = worksheet.Range["B3:D8"][3, 2]; // Cell D6
```

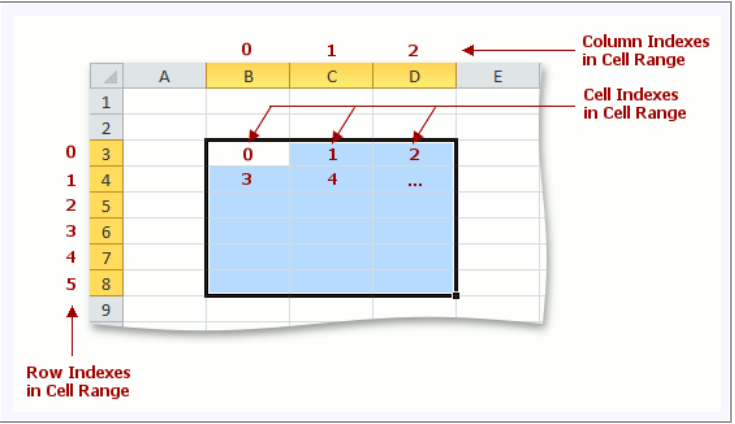
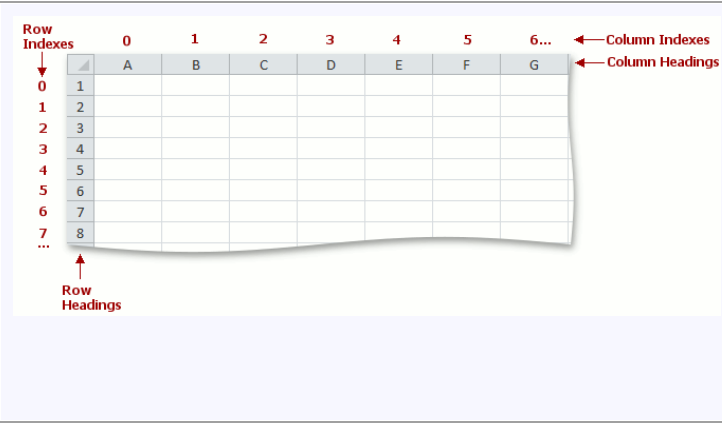
Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
Dim worksheet As Worksheet = workbook.Worksheets(0)
Dim cellA1 As Cell = worksheet(0, 0) ' Cell A1
Dim cellB2 As Cell = worksheet.Cells("B2") ' Cell B2
Dim cellC3 As Cell = worksheet.Cells(2, 2) ' Cell C3
Dim cellD4 As Cell = worksheet.Rows(3)(3) ' Cell D4
Dim cellE5 As Cell = worksheet.Rows(4)("E") ' Cell E5
Dim cellF6 As Cell = worksheet.Rows("6")(5) ' Cell F6
Dim cellG7 As Cell = worksheet.Columns(6)(6) ' Cell G7
Dim cellH8 As Cell = worksheet.Columns("H")(7) ' Cell H8
Dim cellI9 As Cell = worksheet.Columns("I")("9") ' Cell I9
' Access a cell from the range of cells.
Dim cellB5 As Cell = worksheet.Range("B3:D8")(6) ' Cell B5
Dim cellD6 As Cell = worksheet.Range("B3:D8")(3, 2) ' Cell D6
```

The following images demonstrate how rows and columns are indexed in a worksheet and in a specified range of cells (the workbook is opened in Microsoft® Excel®).

Row and Column Indexes and Names in Worksheet

Cell, Row and Column Indexes in a Cell Range



See Also
Cell Basics
[How to: Access a Range of Cells](#)
[How to: Access a Row or Column](#)

How to: Access a Range of Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Access a Range of Cells](#)

This example demonstrates how to access ranges of cells in a worksheet. There are several ways to accomplish this.

- Use the Worksheet.Item property.
- Use the IRangeProvider.Item, IRangeProvider.Parse and IRangeProvider.FromLTRB members of the IRangeProvider object, accessed via the Worksheet.Range or [Workbook.Range](#) property.

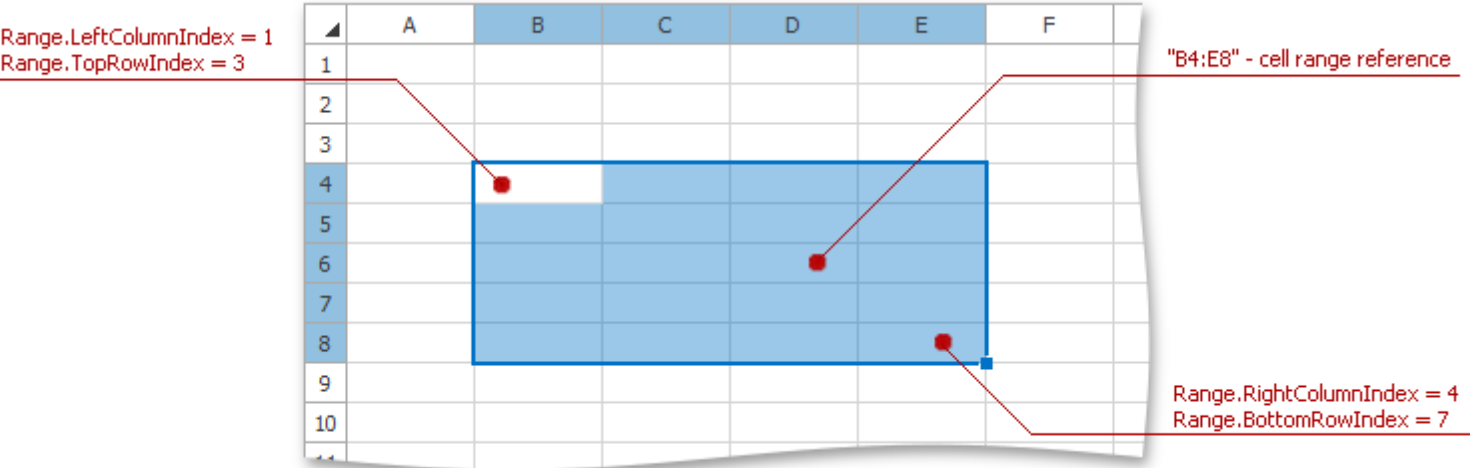
C#

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
Worksheet worksheet = workbook.Worksheets[0];
// A range that includes cells from the top left cell (A1) to the bottom right cell (B5).
Range rangeA1B5 = worksheet["A1:B5"];
// A rectangular range that includes cells from the top left cell (C4) to the bottom right cell (E7).
Range rangeC4E7 = worksheet.Range["C4:E7"];
// The C4:E7 cell range located in the "Sheet3" worksheet.
Range rangeSheet3C4E7 = workbook.Range["Sheet3!C4:E7"];
// A range that contains a single cell (E7).
Range rangeE7 = worksheet.Range["E7"];
// A range that includes the entire column A.
Range rangeColumnA = worksheet.Range["A:A"];
// A range that includes the entire row 5.
Range rangeRow5 = worksheet.Range["5:5"];
// A minimal rectangular range that includes all listed cells: C6, D9 and E7.
Range rangeC6D9E7 = worksheet.Range.Parse("C6:D9:E7");
// A rectangular range whose left column index is 0, top row index is 0,
// right column index is 3 and bottom row index is 2. This is the A1:D3 cell range.
Range rangeA1D3 = worksheet.Range.FromLTRB(0, 0, 3, 2);
// A range that includes the intersection of two ranges: C5:E10 and E9:G13.
// This is the E9:E10 cell range.
Range rangeE9E10 = worksheet.Range["C5:E10 E9:G13"];
// Create a defined name for the D20:G23 cell range.
worksheet.NamedRanges.Add("Range_Name", "Sheet1!$D$20:$G$23");
// Access a range by its defined name.
Range rangeD20G23 = worksheet.Range["Range_Name"];
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
Dim worksheet As Worksheet = workbook.Worksheets(0)
' A range that includes cells from the top left cell (A1) to the bottom right cell (B5).
Dim rangeA1B5 As Range = worksheet.Range("A1:B5")
' A rectangular range that includes cells from the top left cell (C4) to the bottom right cell (E7).
Dim rangeC4E7 As Range = worksheet.Range("C4:E7")
' The C4:E7 cell range located in the "Sheet3" worksheet.
Dim rangeSheet3C4E7 As Range = workbook.Range("Sheet3!C4:E7")
' A range that contains a single cell (E7).
Dim rangeE7 As Range = worksheet.Range("E7")
' A range that includes the entire column A.
Dim rangeColumnA As Range = worksheet.Range("A:A")
' A range that includes the entire row 5.
Dim rangeRow5 As Range = worksheet.Range("5:5")
' A minimal rectangular range that includes all listed cells: C6, D9 and E7.
Dim rangeC6D9E7 As Range = worksheet.Range.Parse("C6:D9:E7")
' A rectangular range whose left column index is 0, top row index is 0,
' right column index is 3 and bottom row index is 2. This is the A1:D3 cell range.
Dim rangeA1D3 As Range = worksheet.Range.FromLTRB(0, 0, 3, 2)
' A range that includes the intersection of two ranges: C5:E10 and E9:G13.
' This is the E9:E10 cell range.
Dim rangeE9E10 As Range = worksheet.Range("C5:E10 E9:G13")
' Create a defined name for the D20:G23 cell range.
worksheet.NamedRanges.Add("Range_Name", "Sheet1!$D$20:$G$23")
' Access a range by its defined name.
Dim rangeD20G23 As Range = worksheet.Range("Range_Name")
```

The following image illustrates a cell range, its top left and bottom right cell indexes, and the cell range reference (the workbook is opened in Microsoft® Excel®).



See Also
[How to: Access a Cell in a Worksheet](#)
[How to: Access a Row or Column](#)

How to: Insert a Cell or Cell Range

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Insert a Cell or Cell Range](#)

This example demonstrates how to insert cells into a worksheet. To do this, use the `Worksheet.InsertCells` method. Pass a [cell](#) or [cell range](#) where you want to insert new cells, and the `InsertCellsMode.ShiftCellsDown` or `InsertCellsMode.ShiftCellsRight` enumeration member to specify how to shift other cells.

C#
<pre>using DevExpress.Spreadsheet; // ... Workbook workbook = new Workbook(); Worksheet worksheet = workbook.Worksheets[0]; // Insert a cell into the C5 position, shifting other cells in the same row to the right. worksheet.InsertCells(worksheet.Cells["C5"], InsertCellsMode.ShiftCellsDown); // Insert cells into the location of the H11:I12 range, shifting other cells in the same column down. worksheet.InsertCells(worksheet.Range["H11:I12"], InsertCellsMode.ShiftCellsRight);</pre>

Visual Basic
<pre>Imports DevExpress.Spreadsheet ' ... Dim workbook As New Workbook() Dim worksheet As Worksheet = workbook.Worksheets(0) ' Insert a cell into the C5 position, shifting other cells in the same row to the right. worksheet.InsertCells(worksheet.Cells("C5"), InsertCellsMode.ShiftCellsDown) ' Insert cells into the location of the H11:I12 range, shifting other cells in the same column down. worksheet.InsertCells(worksheet.Range("H11:I12"), InsertCellsMode.ShiftCellsRight)</pre>

See Also

[How to: Delete a Cell or Range of Cells](#)

[How to: Add a New Row or Column to a Worksheet](#)

How to: Delete a Cell or Range of Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Delete a Cell or Range of Cells](#)

This example demonstrates how to delete cells from a worksheet. To do this, use the `Worksheet.DeleteCells` method. Pass the [cell](#) or [cell range](#) that you wish to delete, and the `DeleteMode.ShiftCellsUp` or `DeleteMode.ShiftCellsLeft` enumeration member to specify how other cells should be shifted.

```
C#
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
Worksheet worksheet = workbook.Worksheets[0];
// Delete the C5 cell, shifting other cells in the same row to the left.
worksheet.DeleteCells(worksheet.Cells["C5"], DeleteMode.ShiftCellsLeft);
// Delete the H11:I12 range of cells, shifting other cells in the same column up.
worksheet.DeleteCells(worksheet.Range["H11:I12"], DeleteMode.ShiftCellsUp);
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Delete the C5 cell, shifting other cells in the same row to the left.
worksheet.DeleteCells(worksheet.Cells("C5"), DeleteMode.ShiftCellsLeft)
' Delete the H11:I12 range of cells, shifting other cells in the same column up.
worksheet.DeleteCells(worksheet.Range("H11:I12"), DeleteMode.ShiftCellsUp)
```

To delete cell content or formatting without removing an entire cell from the worksheet, use the **Clear*** methods of the `Worksheet` object (see the [How to: Clear Cells of Content, Formatting, Hyperlinks and Comments](#) example).

See Also

[How to: Insert a Cell or Cell Range](#)

[How to: Delete a Row or Column from a Worksheet](#)

[How to: Clear Cells of Content, Formatting, Hyperlinks and Comments](#)

How to: Create a Named Range of Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Create a Named Range of Cells](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to create a named range of cells in a worksheet. You can do this in one of the following ways.

- Use the `Worksheet.Range` property to access a cell range (see the [How to: Access a Range of Cells](#) document for details). Then, specify the range name via the `Range.Name` property of the created `Range` object.
- Use the `DefinedNameCollection.Add` method to add a new defined name for the specified range to the worksheet's collection of defined names (`Worksheet.DefinedNames`). Then, use the `Worksheet.Range` property and pass the `DefinedName.Name` property value of the previously created [defined name](#) object.

Note

When specifying a name for a cell or range of cells, follow the rules listed in the [Defined Names](#) document.

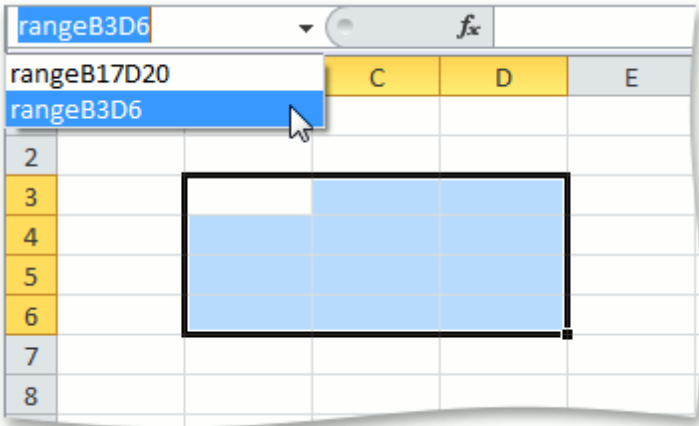
C#

```
(CellActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
// Create a range.
Range rangeB3D6 = worksheet.Range("B3:D6");
// Specify the name for the created range.
rangeB3D6.Name = "rangeB3D6";
// Create a new defined name with the specified range name and absolute reference.
DefinedName definedName = worksheet.DefinedNames.Add("rangeB17D20", "Sheet1!$B$17:$D$20");
// Create a range using the specified defined name.
Range B17D20 = worksheet.Range[definedName.Name];
```

Visual Basic

```
(CellActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Create a range.
Dim rangeB3D6 As Range = worksheet.Range("B3:D6")
' Specify the name for the created range.
rangeB3D6.Name = "rangeB3D6"
' Create a new defined name with the specified range name and absolute reference.
Dim definedName As DefinedName = worksheet.DefinedNames.Add("rangeB17D20", "Sheet1!$B$17:$D$20")
' Create a range using the specified defined name.
Dim B17D20 As Range = worksheet.Range(definedName.Name)
```

The image below shows the defined names of cell ranges in a worksheet (the workbook is opened in Microsoft® Excel®).



To learn how to use named ranges in formulas, see the [How to: Use Names in Formulas](#) document.

See Also

- [How to: Create Named Formulas](#)
- [How to: Use Names in Formulas](#)
- [Defined Names](#)

How to: Change a Cell or Cell Range Value

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Change a Cell or Cell Range Value](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to add data of [different types](#) to worksheet cells. To do this, assign the required value to the Range.Value property of the cell or range object. This property returns the CellValue object, whose properties can be used to obtain the cell value type or get the value itself.

For details on how to access an individual cell or range of cells, refer to the [How to: Access a Cell in a Worksheet](#) or [How to: Access a Range of Cells](#) document.

Note

Note that a [Decimal](#) value can not be assigned to a cell via the Range.Value property. Use the Range.SetValue or CellValue.FromObject method instead.

C#

(CellActions.cs)
// Add data of different types to cells.
worksheet.Cells["B1"].Value = DateTime.Now;
worksheet.Cells["B2"].Value = Math.PI;
worksheet.Cells["B3"].Value = "Have a nice day!";
worksheet.Cells["B4"].Value = CellValue.ErrorReference;
worksheet.Cells["B5"].Value = true;
worksheet.Cells["B6"].Value = float.MaxValue;
worksheet.Cells["B7"].Value = 'a';
worksheet.Cells["B8"].Value = Int32.MaxValue;
// Fill all cells in the range with 10.
worksheet.Range["B10:E10"].Value = 10;

Visual Basic

(CellActions.vb)
' Add data of different types to cells.
worksheet.Cells("B1").Value = Date.Now
worksheet.Cells("B2").Value = Math.PI
worksheet.Cells("B3").Value = "Have a nice day!"
worksheet.Cells("B4").Value = CellValue.ErrorReference
worksheet.Cells("B5").Value = True
worksheet.Cells("B6").Value = Single.MaxValue
worksheet.Cells("B7").Value = "a"
worksheet.Cells("B8").Value = Int32.MaxValue
' Fill all cells in the range with 10.
worksheet.Range("B10:E10").Value = 10

The following image shows how data of different types are displayed in cells (the workbook is opened in Microsoft® Excel®):

	A	B	C	D	E	F
1	dateTime:	4/23/2013				
2	double:	3.141592654				
3	string:	Have a nice day!				
4	error constant:	#REF!				
5	boolean:	TRUE				
6	float:	3.40282E+38				
7	char:	97				
8	int32:	2147483647				
9						
10	Fill a range of cells:	10	10	10	10	
11						

Convert a String to a Cell Value

The example below demonstrates how to use the Range.SetValueFromText method to automatically convert a specified string to a CellValue object of the appropriate [data type](#) and assign it to a cell. Set the method's *preserveNumberFormat* parameter to **true** to retain the cell's Formatting.NumberFormat.

C#

```
// Add data of different types to cells.
worksheet.Cells["B1"].SetValueFromText("28-Jul-20 5:43PM"); // DateTime
worksheet.Cells["B2"].SetValueFromText("3.1415926536"); // double
worksheet.Cells["B3"].SetValueFromText("Have a nice day!"); // string
worksheet.Cells["B4"].SetValueFromText("#REF!"); // error
worksheet.Cells["B5"].SetValueFromText("true"); // Boolean
worksheet.Cells["B6"].SetValueFromText("3.40282E+38"); // float
worksheet.Cells["B7"].SetValueFromText("2147483647"); // int32
worksheet.Cells["B8"].NumberFormat = "d-mmm-yy h:mm";
worksheet.Cells["B8"].SetValueFromText("28-Jul-20 5:43PM", true); // DateTime
worksheet.Cells["B9"].SetValueFromText("=SQRT(25)"); // formula
```

Visual Basic

```
' Add data of different types to cells.
worksheet.Cells("B1").SetValueFromText("28-Jul-20 5:43PM") ' DateTime
worksheet.Cells("B2").SetValueFromText("3.1415926536") ' double
worksheet.Cells("B3").SetValueFromText("Have a nice day!") ' string
worksheet.Cells("B4").SetValueFromText("#REF!") ' error
worksheet.Cells("B5").SetValueFromText("true") ' Boolean
worksheet.Cells("B6").SetValueFromText("3.40282E+38") ' float
worksheet.Cells("B7").SetValueFromText("2147483647") ' int32
worksheet.Cells("B8").NumberFormat = "d-mmm-yy h:mm"
worksheet.Cells("B8").SetValueFromText("28-Jul-20 5:43PM", True) ' DateTime
worksheet.Cells("B9").SetValueFromText("=SQRT(25)") ' formula
```

If you need to assign an object of any type to a cell value, use the Range.SetValue method.

See Also[Cell Data Types](#)[Dates and Times in Cells](#)[Error Types](#)[How to: Specify Number or Date Format for Cell Content](#)

How to: Add Formulas to Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Add Formulas to Cells](#)

To add a [formula](#) to a cell, use the Range.Formula property of the Cell object. Assign the required formula to this property as a string starting with the "=" sign. For more information on formulas, refer to the [Spreadsheet Formulas](#) section, and review the following examples.

- [How to: Use Constants and Calculation Operators in Formulas](#)
- [How to: Use Cell and Worksheet References in Formulas](#)
- [How to: Use Names in Formulas](#)
- [How to: Use Functions and Nested Functions in Formulas](#)

When you set the Range.Formula property for the [cell range](#), each cell in this range will contain the specified formula with adjusted cell references (all relative cell references included in the formula will automatically be changed). This approach allows you to avoid duplicating the same formula for multiple cells and manually updating cell references. Refer to the [How to: Create Shared Formulas](#) example for more information.

To perform calculations with arrays of cells, use [array formulas](#). To create an array formula, call the Range.ArrayFormula method. Refer to the [How to: Create Array Formulas](#) example for more information.

See Also

[Spreadsheet Formulas](#)

How to: Add a Hyperlink to a Cell

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Add a Hyperlink to a Cell](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to create a hyperlink to a web page or range in a workbook. To do this, use the `HyperlinkCollection.Add` method with the passed cell or cell range into which a hyperlink should be inserted, a target web page or workbook location and other parameters.

All hyperlinks created in a worksheet are contained in the `HyperlinkCollection` collection returned by the `Worksheet.Hyperlinks` property. To adjust an existing hyperlink, use properties of the `Hyperlink` object that can be accessed by the hyperlink index from the `HyperlinkCollection` collection. To get all hyperlinks contained in the specified cell range, use the `HyperlinkCollection.GetHyperlinks` method.

To remove hyperlinks, use the `HyperlinkCollection.Remove` or `HyperlinkCollection.RemoveAt` method. The `Worksheet.ClearHyperlinks` method deletes all hyperlinks from the specified range of cells.

C#

```
(CellActions.cs)
// Create a hyperlink to a web page.
Cell cell = worksheet.Cells["A1"];
worksheet.Hyperlinks.Add(cell, "http://www.devexpress.com/", true, "DevExpress");
// Create a hyperlink to a cell range in a workbook.
Range range = worksheet.Range["C3:D4"];
Hyperlink cellHyperlink = worksheet.Hyperlinks.Add(range, "Sheet2!B2:E7", false, "Select Range");
cellHyperlink.TooltipText = "Click Me";
```

Visual Basic

```
(CellActions.vb)
' Create a hyperlink to a web page.
Dim cell As Cell = worksheet.Cells("A1")
worksheet.Hyperlinks.Add(cell, "http://www.devexpress.com/", True, "DevExp
' Create a hyperlink to a cell range in a workbook.
Dim range As Range = worksheet.Range("C3:D4")
Dim cellHyperlink As Hyperlink = worksheet.Hyperlinks.Add(range, "Sheet2!B
cellHyperlink.TooltipText = "Click Me"
```

How To: Add a Comment To a Cell

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How To: Add a Comment To a Cell](#)

This example demonstrates how to add a [comment](#) to a cell and format the comment text.

1. To create a new comment and associate it with a cell, access the worksheet's collection of cell comments from the `Worksheet.Comments` property and call the `CommentCollection.Add` method. Pass the following parameters:
 - a `Cell` object that specifies the [cell](#) to be commented;
 - a string that specifies the author of the comment. In this example, the system username is used. Access it from the [Workbook.CurrentAuthor](#) property;
 - a string that specifies the text of the comment.
2. To apply different fonts to specific regions of the comment text, modify the `CommentRunCollection` collection, which is accessed from the `Comment.Runs` property. This collection stores the [CommentRun](#) objects ([comment runs](#)) that define regions of the comment text that are formatted specifically. After a comment has been created, its text is defined by a single run that is contained in the `CommentRunCollection` collection.

Add more runs to the comment.

- Insert the author's name at the beginning of the comment text (using the `CommentRunCollection.Insert` method) and format it as bold.
- Access the comment text that was added initially. It is now defined by the second run in the collection of the comment runs. Modify font characteristics for this text region.
- Add one more run to the end of the comment text using the `CommentRunCollection.Add` method.

As you can see, cell comments can be formatted using methods of the `CommentRunCollection` object.

To remove comments from cells, use the `CommentCollection.Remove`, `CommentCollection.RemoveAt`, `CommentCollection.Clear` or `Worksheet.ClearComments` methods.

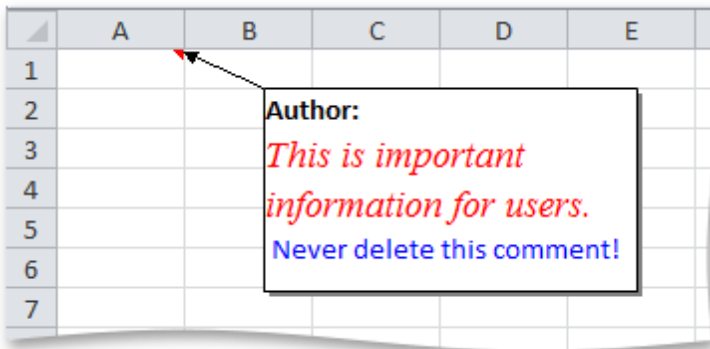
C#

```
using DevExpress.Spreadsheet;
//...
Workbook workbook = new Workbook();
Worksheet worksheet = workbook.Worksheets[0];
//Get the system username.
string author = workbook.CurrentAuthor;
//Add a comment to the A1 cell.
Cell cell = worksheet.Cells["A1"];
Comment comment = worksheet.Comments.Add(cell, author, "This is important information for users.");
//Add the author name at the beginning of the comment.
CommentRunCollection runs = comment.Runs;
runs.Insert(0, author + ": \r\n");
runs[0].Font.Bold = true;
//Format the comment text.
runs[1].Font.Color = Color.Red;
runs[1].Font.Name = "Times New Roman";
runs[1].Font.Size = 14;
runs[1].Font.Italic = true;
//Add a new comment run.
runs.Add("\n Never delete this comment!");
runs[2].Font.Color = Color.MidnightBlue;
```

Visual Basic

```
Imports DevExpress.Spreadsheet
'...
Dim workbook As New Workbook()
Dim worksheet As Worksheet = workbook.Worksheets(0)
'Get the system username.
Dim author As String = workbook.CurrentAuthor
'Add a comment to the A1 cell.
Dim cell As Cell = worksheet.Cells("A1")
Dim comment As Comment = worksheet.Comments.Add(cell, author, "This is important information for users.")
'Add the author name at the beginning of the comment.
Dim runs As CommentRunCollection = comment.Runs
runs.Insert(0, author & ": " & Constants.vbCrLf)
runs(0).Font.Bold = True
'Format the comment text.
runs(1).Font.Color = Color.Red
runs(1).Font.Name = "Times New Roman"
runs(1).Font.Size = 14
runs(1).Font.Italic = True
'Add a new comment run.
runs.Add(Constants.vbLf & "Never delete this comment!")
runs(2).Font.Color = Color.MidnightBlue
```

The image below shows the comment (the workbook is opened in Microsoft® Excel®).



How to: Clear Cells of Content, Formatting, Hyperlinks and Comments

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Clear Cells of Content, Formatting, Hyperlinks and Comments](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to clear worksheet cells.

- **Clear All**

To remove all cell information (content, formatting, hyperlinks and comments), call the `Worksheet.Clear` method.

- **Clear Cell Content**

To remove cell content only ([value](#) and [formula](#)), call the `Worksheet.ClearContents` method, or assign the `Range.Value` property to **null** or to `CellValue.Empty`.

- **Clear Cell Formatting**

To remove cell formatting only, call the `Worksheet.ClearFormats` method, or apply the *Normal* style to cells via the `Range.Style` property.

- **Clear Cell Hyperlinks**

To remove any [hyperlinks](#) contained in a cell or cell range, call the `Worksheet.ClearHyperlinks` method.

You can also get hyperlinks contained in the specified cell range via the `HyperlinkCollection.GetHyperlinks` method and remove them using the `HyperlinkCollection.Remove` method of the `Worksheet.Hyperlinks` collection. In this case, cell formatting is also cleared.

- **Clear Cell Comments**

To remove any comments contained in a cell or cell range, call the `Worksheet.ClearComments` method, or remove comments from the worksheet's `Worksheet.Comments` collection via the `CommentCollection.Remove` or `CommentCollection.RemoveAt` method.

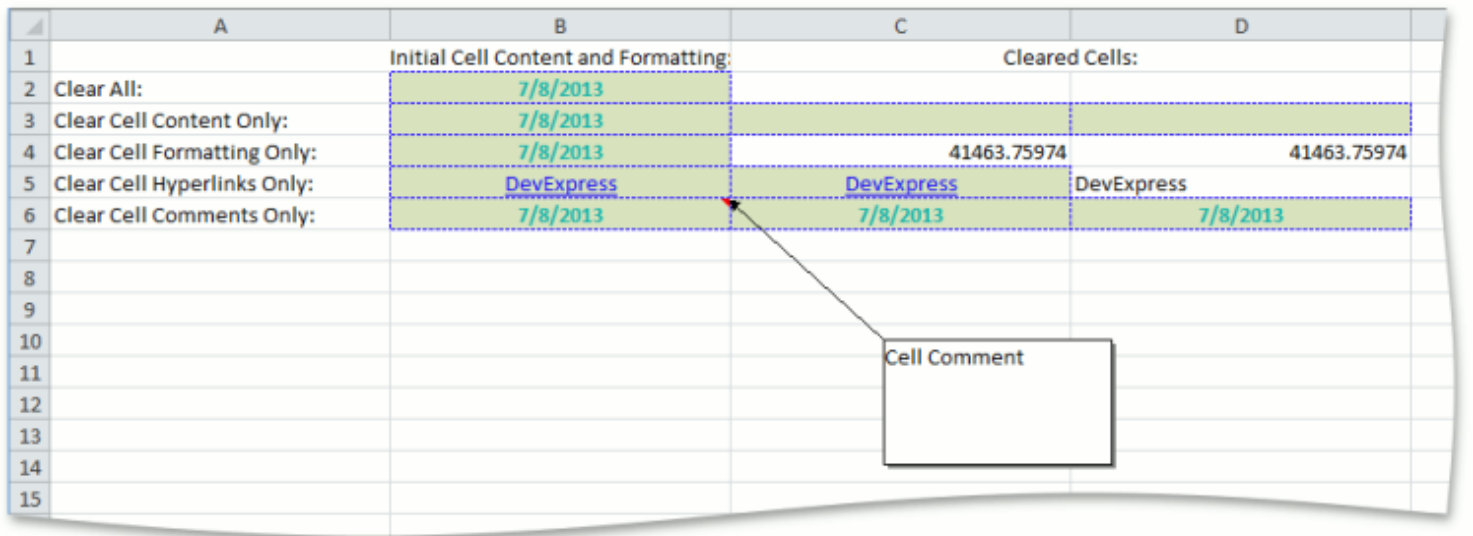
C#

```
(CellActions.cs)
// Remove all cell information (content, formatting, hyperlinks and comments).
worksheet.Clear(worksheet["C2:D2"]);
// Remove cell content.
worksheet.ClearContents(worksheet["C3"]);
worksheet["D3"].Value = null;
// Remove cell formatting.
worksheet.ClearFormats(worksheet["C4"]);
worksheet["D4"].Style = workbook.Styles.DefaultStyle;
// Remove hyperlinks from cells.
worksheet.ClearHyperlinks(worksheet["C5"]);
Hyperlink hyperlinkD5 = worksheet.Hyperlinks.GetHyperlinks(worksheet["D5"])[0];
worksheet.Hyperlinks.Remove(hyperlinkD5);
// Remove comments from cells.
worksheet.ClearComments(worksheet["C6"]);
Comment commentD6 = worksheet.Comments.GetComments(worksheet["D6"])[0];
worksheet.Comments.Remove(commentD6);
```

Visual Basic

```
(CellActions.vb)
' Remove all cell information (content, formatting, hyperlinks and comment
worksheet.Clear(worksheet("C2:D2"))
' Remove cell content.
worksheet.ClearContents(worksheet("C3"))
worksheet("D3").Value = Nothing
' Remove cell formatting.
worksheet.ClearFormats(worksheet("C4"))
worksheet("D4").Style = workbook.Styles.DefaultStyle
' Remove hyperlinks from cells.
worksheet.ClearHyperlinks(worksheet("C5"))
Dim hyperlinkD5 As Hyperlink = worksheet.Hyperlinks.GetHyperlinks(worksheet("D5"))
worksheet.Hyperlinks.Remove(hyperlinkD5)
' Remove comments from cells.
worksheet.ClearComments(worksheet("C6"))
Dim commentD6 As Comment = worksheet.Comments.GetComments(worksheet("D6"))
worksheet.Comments.Remove(commentD6)
```

The image below shows how cells can be cleared (the workbook is opened in Microsoft® Excel®).



To delete an entire cell or range of cells from the worksheet, use the Worksheet.DeleteCells method (see the [How to: Delete a Cell or Range of Cells](#) example).

See Also
[How to: Delete a Cell or Range of Cells](#)

How to: Copy Cell Data Only, Cell Style Only, or Cell Data with Style

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Copy Cell Data Only, Cell Style Only, or Cell Data with Style](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to copy data from one cell to another, and specify which parts of the copied data (for example, value only, formatting information only, borders only, etc.) should be pasted into the destination cell. To do this, call the Range.CopyFrom method of the Cell object corresponding to the cell into which you want to copy data from another cell. Pass the source cell object and required member of the PasteSpecial enumerator as parameters.

C#

(CellActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
worksheet.Columns["A"].WidthInCharacters = 32;
worksheet.Columns["B"].WidthInCharacters = 20;
Style style = workbook.Styles[BuiltInStyleId.Input];
// Specify the content and formatting for a source cell.
worksheet.Cells["A1"].Value = "Source Cell";
Cell sourceCell = worksheet.Cells["B1"];
sourceCell.Formula = "= PI()";
sourceCell.NumberFormat = "0.0000";
sourceCell.Style = style;
sourceCell.Font.Color = Color.Blue;
sourceCell.Font.Bold = true;
sourceCell.Borders.SetOutsideBorders(Color.Black, BorderLineStyle.Thin);
// Copy all information from the source cell to the "B3" cell.
worksheet.Cells["A3"].Value = "Copy All";
worksheet.Cells["B3"].CopyFrom(sourceCell);
// Copy only the source cell content (e.g., text, numbers, formula calculated values) to the "B4" cell.
worksheet.Cells["A4"].Value = "Copy Values";
worksheet.Cells["B4"].CopyFrom(sourceCell, PasteSpecial.Values);
// Copy the source cell content (e.g., text, numbers, formula calculated values)
// and number formats to the "B5" cell.
worksheet.Cells["A5"].Value = "Copy Values and Number Formats";
worksheet.Cells["B5"].CopyFrom(sourceCell, PasteSpecial.Values | PasteSpecial.NumberFormats);
// Copy only the formatting information from the source cell to the "B6" cell.
worksheet.Cells["A6"].Value = "Copy Formats";
worksheet.Cells["B6"].CopyFrom(sourceCell, PasteSpecial.Formats);
// Copy all information from the source cell to the "B7" cell except for border settings.
worksheet.Cells["A7"].Value = "Copy All Except Borders";
worksheet.Cells["B7"].CopyFrom(sourceCell, PasteSpecial.All & ~PasteSpecial.Borders);
// Copy information only about borders from the source cell to the "B8" cell.
worksheet.Cells["A8"].Value = "Copy Borders";
worksheet.Cells["B8"].CopyFrom(sourceCell, PasteSpecial.Borders);

Visual Basic

```

(CellActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets(0)
worksheet.Columns("A").WidthInCharacters = 32
worksheet.Columns("B").WidthInCharacters = 20
Dim style As Style = workbook.Styles(BuiltInStyleId.Input)
' Specify the content and formatting for a source cell.
worksheet.Cells("A1").Value = "Source Cell"
Dim sourceCell As Cell = worksheet.Cells("B1")
sourceCell.Formula = "= PI()"
sourceCell.NumberFormat = "0.0000"
sourceCell.Style = style
sourceCell.Font.Color = Color.Blue
sourceCell.Font.Bold = True
sourceCell.Borders.SetOutsideBorders(Color.Black, BorderLineStyle.Thin)
' Copy all information from the source cell to the "B3" cell.
worksheet.Cells("A3").Value = "Copy All"
worksheet.Cells("B3").CopyFrom(sourceCell)
' Copy only the source cell content (e.g., text, numbers, formula calculated va
worksheet.Cells("A4").Value = "Copy Values"
worksheet.Cells("B4").CopyFrom(sourceCell, PasteSpecial.Values)
' Copy the source cell content (e.g., text, numbers, formula calculated va
' and number formats to the "B5" cell.
worksheet.Cells("A5").Value = "Copy Values and Number Formats"
worksheet.Cells("B5").CopyFrom(sourceCell, PasteSpecial.Values Or PasteSpe
' Copy only the formatting information from the source cell to the "B6" ce
worksheet.Cells("A6").Value = "Copy Formats"
worksheet.Cells("B6").CopyFrom(sourceCell, PasteSpecial.Formats)
' Copy all information from the source cell to the "B7" cell except for bo
worksheet.Cells("A7").Value = "Copy All Except Borders"
worksheet.Cells("B7").CopyFrom(sourceCell, PasteSpecial.All And (Not Paste
' Copy information only about borders from the source cell to the "B8" cel
worksheet.Cells("A8").Value = "Copy Borders"
worksheet.Cells("B8").CopyFrom(sourceCell, PasteSpecial.Borders)

```

The image below shows how different parts of cell data can be copied and pasted into other cells (the workbook is opened in Microsoft® Excel®).

	A	B	C
1	Source Cell	3.1416	
2			
3	Copy All	3.1416	
4	Copy Values	3.141592654	
5	Copy Values and Number Formats	3.1416	
6	Copy Formats		
7	Copy All Except Borders	3.1416	
8	Copy Borders		
9			

How to: Merge Cells or Split Merged Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Cells](#) > [How to: Merge Cells or Split Merged Cells](#)

• Merge Cells

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to merge several cells into a single cell. To do this, use the `Worksheet.MergeCells` method with the passed [range of cells](#) to be merged.

Note

When you merge a cell range, the data (value, formula and format settings) of only the top left non-empty cell will appear in the resulting merged cell. The data contained in other cells of the original range will be lost.

If all cells within the original range are empty, the range's top left cell format will be applied to the merged cell.

C#

```
(CellActions.cs)
// Merge cells contained in the range.
worksheet.MergeCells(worksheet.Range["A1:C5"]);
```

Visual Basic

```
(CellActions.vb)
' Merge cells contained in the range.
worksheet.MergeCells(worksheet.Range("A1:C5"))
```

Split Merged Cells

Call the `Worksheet.UnMergeCells` method with the passed range that contains merged cells to be split. Note that the content and formatting that were set in cells before merging will be lost (with the exception of the top left cell).

C#

```
// Split cells that were previously merged.
worksheet.UnMergeCells(worksheet.Range["A1:C5"]);
```

Visual Basic

```
' Split cells that were previously merged.
worksheet.UnMergeCells(worksheet.Range("A1:C5"))
```

Split All Merged Cells in a Worksheet

To get all merged cells in the specified cell range, use the `Range.GetMergedRanges` method.

C#

```
// Split all merged cells in the worksheet.
foreach (var item in worksheet.Cells.GetMergedRanges()) {
    item.UnMerge();
}
```

Visual Basic

```
' Split all merged cells in the worksheet.  
For Each item In worksheet.Cells.GetMergedRanges()  
    item.UnMerge()  
Next item
```

The image below shows how cells are merged and split in a worksheet (the workbook is opened in Microsoft® Excel®). Note that only the "B2" data (the top left non-empty cell) appears in the merged cell.

	A	B	C	D
1				
2		B2		
3			C3	
4				
5				
6				
7				

worksheet.MergeCells(worksheet.CreateRange("A1:C5"))



	A	B	C	D
1	B2			
2				
3				
4				
5				
6				
7				

worksheet.UnMergeCells(worksheet.CreateRange("A1:C5"))



	A	B	C	D
1	B2			
2				
3				
4				
5				
6				
7				

Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formulas](#)

The following examples demonstrate how to create and use different [formulas](#) in cells.

- [How to: Use Constants and Calculation Operators in Formulas](#)
- [How to: Use Cell and Worksheet References in Formulas](#)
- [How to: Use Names in Formulas](#)
- [How to: Create Named Formulas](#)
- [How to: Use Functions and Nested Functions in Formulas](#)
- [How to: Create Shared Formulas](#)
- [How to: Create Array Formulas](#)

See Also

[Spreadsheet Formulas](#)

How to: Use Constants and Calculation Operators in Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formulas](#) > [How to: Use Constants and Calculation Operators in Formulas](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to create a simple [formula](#) using constants and calculation [operators](#), and add this formula to a cell. Constants are values that are not calculated (e.g., date values, numbers, text strings). Operators specify what calculations should be performed on a formula's elements.

C#

```
(FormulaActions.cs)
// Use constants and calculation operators in a formula.
workbook.Worksheets[0].Cells["B2"].Formula = "= (1+5)*6";
```

Visual Basic

```
(FormulaActions.vb)
' Use constants and calculation operators in a formula.
workbook.Worksheets(0).Cells("B2").Formula = "= (1+5)*6"
```

See Also

[Spreadsheet Formulas](#)

[Operators](#)

[How to: Use Cell and Worksheet References in Formulas](#)

[How to: Use Names in Formulas](#)

[How to: Use Functions and Nested Functions in Formulas](#)

How to: Use Cell and Worksheet References in Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formulas](#) > [How to: Use Cell and Worksheet References in Formulas](#)

In addition to [constants](#), a formula can contain references to other cells or cell ranges in the same worksheet or other worksheets. The following reference styles are supported.

- [A1 References](#)
- [Cross-Worksheet References](#)
- [Relative References](#)
- [Absolute References](#)
- [Mixed References](#)
- [3D References](#)
- [R1C1 References](#)

• A1 References

A cell reference of the A1 style is a combination of column and row headings to which the cell belongs - a column letter is followed by a row number. For example, C10 refers to the cell that is located at the intersection of column C and row 10. By default, column and row headings are displayed at the top and at the left of a worksheet (see [How to: Show and Hide Row and Column Headings](#)).

C#

```
// Use the A1 reference style in a formula.
workbook.Worksheets[0].Cells["A1"].Formula = "= B1+C1";
```

Visual Basic

```
' Use the A1 reference style in a formula.
workbook.Worksheets(0).Cells("A1").Formula = "= B1+C1"
```

• Cross-Worksheet References

In formulas, you can also use references to cells located in other worksheets. To do this, specify the worksheet name before the cell reference, and separate them by an exclamation point (!).

C#

```
// Sum values of cells located in different worksheets.
workbook.Worksheets["Sheet1"].Cells["H15"].Formula = "= Sheet2!C3 + Sheet3!C5";
```

Visual Basic

```
' Sum values of cells located in different worksheets.
workbook.Worksheets("Sheet1").Cells("H15").Formula = "= Sheet2!C3 + Sheet3!C5"
```

• Relative References

A relative cell reference in a formula is based on the relative position of a referenced cell and a cell containing a formula. To keep this relative position unchanged, the cell reference is automatically changed each time you copy or move a formula to another cell. For example, if you copy a formula with a relative reference to cell A1 from cell B2 to C3, the reference will automatically change from A1 to B2.

C#

```
// Use a relative cell reference in a formula.
workbook.Worksheets[0].Cells["B2"].Formula = "= A1";
```

Visual Basic

```
' Use a relative cell reference in a formula.
workbook.Worksheets(0).Cells("B2").Formula = "= A1"
```

• Absolute References

An absolute cell reference in a formula always refers to a specific cell, and it does not change if the formula is copied or moved to another cell. In absolute cell references, the column letter and row number are preceded by the '\$' sign. For example, if you copy a formula with an absolute reference to cell A1 from cell C3 to D4, the reference will remain unchanged (\$A\$1).

C#

```
// Use an absolute cell reference in a formula.
workbook.Worksheets[0].Cells["C3"].Formula = "= $A$1";
```

Visual Basic

```
' Use an absolute cell reference in a formula.
workbook.Worksheets(0).Cells("C3").Formula = "= $A$1"
```

• Mixed References

A mixed cell reference in a formula can either be combined from an absolute reference to a cell column and a relative reference to a cell row (for example, \$B2), or from a relative reference to a cell column and an absolute reference to a cell row (for example, A\$1). If the formula is copied or moved, the absolute element of the mixed reference (the column letter or row number preceded by the '\$' sign) will remain unchanged and the relative element of the reference will automatically be adjusted. For example, if you copy a formula with the \$B2 mixed reference from cell E5 to F6, the reference will change to \$B3.

C#

```
// Use mixed cell references in formulas.
workbook.Worksheets[0].Cells["D4"].Formula = "= A$1";
workbook.Worksheets[0].Cells["E5"].Formula = "= $B2";
```

Visual Basic

```
' Use mixed cell references in formulas.
workbook.Worksheets(0).Cells("D4").Formula = "= A$1"
workbook.Worksheets(0).Cells("E5").Formula = "= $B2"
```

• 3D References

3D references allow you to process data contained in the same cells on multiple worksheets within a workbook. To create a 3D reference, specify the range of worksheet names before the cell (or cell range) reference, and separate them by an exclamation point (!).

C#

```
// Use 3D cell references in formulas.
workbook.Worksheets[0].Cells["G10"].Formula = "= Sheet1!C3";
workbook.Worksheets[0].Cells["G11"].Formula = "= SUM(Sheet1:Sheet3!C3:C5)";
```

Visual Basic

```
' Use 3D cell references in formulas.
workbook.Worksheets(0).Cells("G10").Formula = "= Sheet1!C3"
workbook.Worksheets(0).Cells("G11").Formula = "= SUM(Sheet1:Sheet3!C3:C5) "
```

• R1C1 References

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to create formulas using the [R1C1 reference style](#). To do this, switch on the DocumentSettings.R1C1ReferenceStyle option and assign a formula string containing R1C1 cell references to the Range.Formula property. To obtain a cell reference in the R1C1 reference style, you can use the Range.GetReferenceR1C1 method.

C#

```
(FormulaActions.cs)
// Switch on the R1C1 reference style in a workbook.
workbook.DocumentSettings.R1C1ReferenceStyle = true;
// Specify a formula with relative R1C1 references in cell D2
// to add values contained in cells A2 through A11.
worksheet.Cells["D2"].Formula = "=SUM(RC[-3]:R[9]C[-3])";
// Specify a formula with absolute R1C1 references
// to add values contained in cells A2 through A11.
worksheet.Cells["D3"].Formula = "=SUM(R2C1:R11C1)";
```

Visual Basic

```
(FormulaActions.vb)
' Switch on the R1C1 reference style in a workbook.
workbook.DocumentSettings.R1C1ReferenceStyle = True
' Specify a formula with relative R1C1 references in cell D2
' to add values contained in cells A2 through A11.
worksheet.Cells("D2").Formula = "=SUM(RC[-3]:R[9]C[-3])"
' Specify a formula with absolute R1C1 references
' to add values contained in cells A2 through A11.
worksheet.Cells("D3").Formula = "=SUM(R2C1:R11C1)"
```

How to: Use Names in Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formulas](#) > [How to: Use Names in Formulas](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to use a [named cell range](#) in a [formula](#). In some cases, it can be useful to name individual cells and cell ranges to make it easier to understand what information they contain.

C#

(FormulaActions.cs)
// Access the "A2:C5" range of cells in the worksheet.
Range range = worksheet.Range["A2:C5"];
// Specify the name for the created range.
range.Name = "myRange";
// Create a formula that sums up the values of all cells included in the specified named range.
worksheet.Cells["F3"].Formula = "= SUM(myRange)";

Visual Basic

(FormulaActions.vb)
' Access the "A2:C5" range of cells in the worksheet.
Dim range As Range = worksheet.Range("A2:C5")
' Specify the name for the created range.
range.Name = "myRange"
' Create a formula that sums up the values of all cells included in the specified named range.
worksheet.Cells("F3").Formula = "= SUM(myRange)";

The image below shows a named cell range, and the result returned by the formula using this range (the workbook is opened in Microsoft® Excel®).

myRange				fx 2			
	A	B	C	D	E	F	G
1	myRange:				Sum:		
2	2	2	2		Formula:	= SUM(myRange)	
3	2	2	2		Value:	24	
4	2	2	2				
5	2	2	2				
6							
7							

See Also
[How to: Create a Named Range of Cells](#)
[How to: Create Named Formulas](#)
[Defined Names](#)

How to: Create Named Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formulas](#) > [How to: Create Named Formulas](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to define names for [formulas](#). To do this, call the `DefinedNameCollection.Add` method with a name to be associated with a formula and the formula string passed as parameters. Use the `Worksheet.DefinedNames` or `Workbook.DefinedNames` property to access and modify the collection of defined names of a particular worksheet or entire workbook, depending on which scope you want to specify for a name.

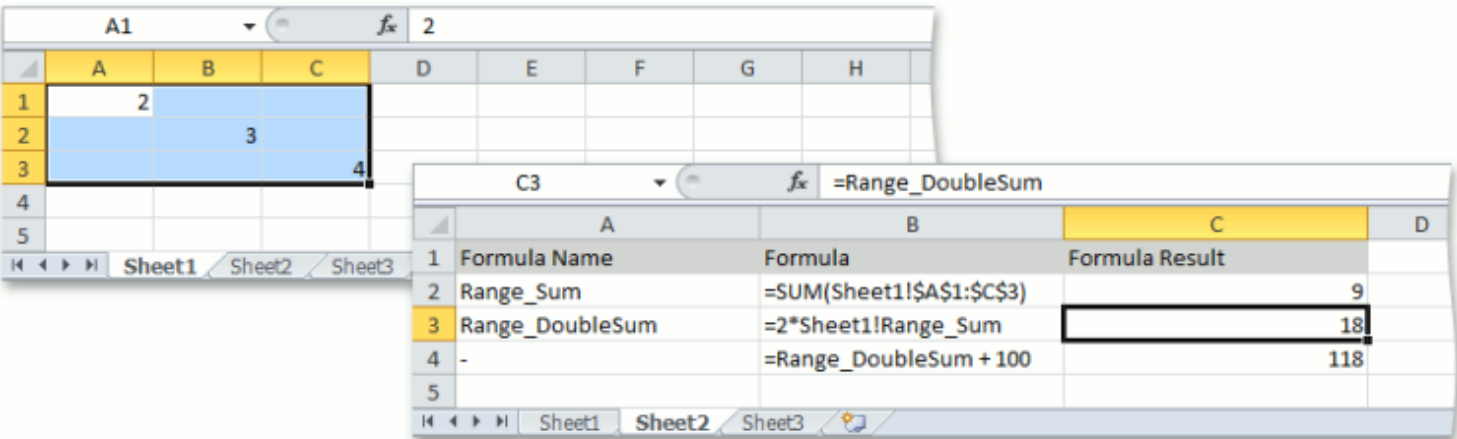
C#

(FormulaActions.cs)
Worksheet worksheet1 = workbook.Worksheets["Sheet1"];
Worksheet worksheet2 = workbook.Worksheets["Sheet2"];
// Create a name for a formula that sums up the values of all cells included in the "A1:C3" range of the '
// The scope of this name will be limited by the "Sheet1" worksheet.
worksheet1.DefinedNames.Add("Range_Sum", "=SUM(Sheet1!\$A\$1:\$C\$3)");
// Create a name for a formula that doubles the value resulting from the "Range_Sum" named formula and
// make this name available within the entire workbook.
workbook.DefinedNames.Add("Range_DoubleSum", "=2*Sheet1!Range_Sum");
// Create formulas that use other formulas with the specified names.
worksheet2.Cells["C2"].Formula = "=Sheet1!Range_Sum";
worksheet2.Cells["C3"].Formula = "=Range_DoubleSum";
worksheet2.Cells["C4"].Formula = "=Range_DoubleSum + 100";

Visual Basic

(FormulaActions.vb)
Dim worksheet1 As Worksheet = workbook.Worksheets("Sheet1")
Dim worksheet2 As Worksheet = workbook.Worksheets("Sheet2")
' Create a name for a formula that sums up the values of all cells included
' The scope of this name will be limited by the "Sheet1" worksheet.
worksheet1.DefinedNames.Add("Range_Sum", "=SUM(Sheet1!\$A\$1:\$C\$3)")
' Create a name for a formula that doubles the value resulting from the "R
' make this name available within the entire workbook.
workbook.DefinedNames.Add("Range_DoubleSum", "=2*Sheet1!Range_Sum")
' Create formulas that use other formulas with the specified names.
worksheet2.Cells("C2").Formula = "=Sheet1!Range_Sum"
worksheet2.Cells("C3").Formula = "=Range_DoubleSum"
worksheet2.Cells("C4").Formula = "=Range_DoubleSum + 100"

The image below shows how to use named formulas in worksheet cells (the workbook is opened in Microsoft® Excel®).



See Also

- [How to: Create a Named Range of Cells](#)
- [How to: Use Names in Formulas](#)
- [Defined Names](#)

How to: Use Functions and Nested Functions in Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formulas](#) > [How to: Use Functions and Nested Functions in Formulas](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to use predefined [functions](#) in [formulas](#) to perform simple or complex calculations over arguments. Set the Range.Formula property to a formula. Follow the rules below to create a formula that uses a function.

1. Start a formula with the "=" sign, as you usually do when creating a formula.
2. Type the function name.
3. Type the function arguments in parentheses. Arguments can be numbers, text and logical values, cell references and names, or other functions.

C#

```
(FormulaActions.cs)
// If the number in cell A2 is less than 10, the formula returns "Normal"
// and this text is displayed in cell C2. Otherwise, cell C2 displays "Excess".
worksheet.Cells["C2"].Formula = @"=IF(A2<10, ""Normal"", ""Excess"");
// Calculate the average value for cell values within the "A2:A7" range.
worksheet.Cells["C3"].Formula = "=AVERAGE(A2:A7)";
// Add the values contained in cells A3 through A5, add the value contained in cell A6,
// and add 100 to that result.
worksheet.Cells["C4"].Formula = "=SUM(A3:A5,A6,100)";
// Use a nested function in a formula.
// Round the sum of the values contained in cells A6 and A7 to two decimal places.
worksheet.Cells["C5"].Formula = "=ROUND(SUM(A6,A7),2)";
// Add the current date to cell C6.
worksheet.Cells["C6"].Formula = "=Today()";
worksheet.Cells["C6"].NumberFormat = "m/d/yy";
// Convert the specified text to uppercase.
worksheet.Cells["C7"].Formula = @"=UPPER(""formula"");"
```

Visual Basic

```
(FormulaActions.vb)
' If the number in cell A2 is less than 10, the formula returns "Normal"
' and this text is displayed in cell C2. Otherwise, cell C2 displays "Exce
worksheet.Cells("C2").Formula = "=IF(A2<10, ""Normal"", ""Excess"")"
' Calculate the average value for cell values within the "A2:A7" range.
worksheet.Cells("C3").Formula = "=AVERAGE(A2:A7)"
' Add the values contained in cells A3 through A5, add the value contained
' and add 100 to that result.
worksheet.Cells("C4").Formula = "=SUM(A3:A5,A6,100)"
' Use a nested function in a formula.
' Round the sum of the values contained in cells A6 and A7 to two decimal
worksheet.Cells("C5").Formula = "=ROUND(SUM(A6,A7),2)"
' Add the current date to cell C6.
worksheet.Cells("C6").Formula = "=Today()"
worksheet.Cells("C6").NumberFormat = "m/d/yy"
' Convert the specified text to uppercase.
worksheet.Cells("C7").Formula = "=UPPER(""formula"")"
```

The image below shows cells with formulas using different functions (the workbook is opened in Microsoft® Excel®).

C5

f_x

=ROUND(SUM(A6,A7),2)

	A	B	C	D
1	Data	Formula String	Formula	
2	15	= IF(A2<10, "Normal", "Excess")	Excess	
3	3	=AVERAGE(A2:A7)	9.853908333	
4	3	=SUM(A3:A5,A6,100)	129	
5	3	=ROUND(SUM(A6,A7),2)	35.12	
6	20	= Today()	5/13/2013	
7	15.12345	=UPPER("formula")	FORMULA	
8				

See Also
[Spreadsheet Formulas](#)
[Functions](#)

How to: Create Shared Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formulas](#) > [How to: Create Shared Formulas](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to apply a shared formula in a range of cells to avoid duplicating the same formula for each cell in the range, and reduce the size of the spreadsheet file. To do this, assign the formula string to the Range.Formula property of the cell range. The shared formula will be created and applied automatically.

C#

(FormulaActions.cs)
worksheet.Cells["A2"].Value = 1;
// Use the shared formula in the "A3:A11" range of cells.
worksheet.Range["A3:A11"].Formula = "=SUM(A2+1)";
// Use the shared formula in the "B2:B11" range of cells.
worksheet.Range["B2:B11"].Formula = "=A2+2";

Visual Basic

(FormulaActions.vb)
worksheet.Cells("A2").Value = 1
' Use the shared formula in the "A3:A11" range of cells.
worksheet.Range("A3:A11").Formula = "=SUM(A2+1) "
' Use the shared formula in the "B2:B11" range of cells.
worksheet.Range("B2:B11").Formula = "=A2+2 "

The following image shows the result of executing the code above (the workbook is opened in Microsoft® Excel®).

	A	B	C	D
1				
2	1	3		
3	2	4		
4	3	5		
5	4	6		
6	5	7		
7	6	8		
8	7	9		
9	8	10		
10	9	11		
11	10	12		
12				
13				

See Also
[Spreadsheet Formulas](#)

How to: Create Array Formulas

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formulas](#) > [How to: Create Array Formulas](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to create an [array formula](#).

C#

```
(FormulaActions.cs)
// Create an array formula that multiplies values contained in the cell range A2 through A11
// by the corresponding cells in the range B2 through B11,
// and displays the results in cells C2 through C11.
worksheet.Range["C2:C11"].ArrayFormula = "=A2:A11*B2:B11";
// Create an array formula that multiplies values contained in the cell range C2 through C11 by 2
// and displays the results in cells D2 through D11.
worksheet.Range["D2:D11"].ArrayFormula = "=C2:C11*2";
// Create an array formula that multiplies values contained in the cell range B2 through D11,
// adds the results, and displays the total sum in cell D12.
worksheet.Cells["D12"].ArrayFormula = "=SUM(B2:B11*C2:C11*D2:D11)";
// Re-dimension an array formula range:
// delete the array formula and create a new range with the same formula.
if (worksheet.Cells["C13"].HasArrayFormula) {
    string af = worksheet.Cells["C13"].ArrayFormula;
    worksheet.Cells["C13"].GetArrayFormulaRange().ArrayFormula = string.Empty;
    worksheet.Range["C2:C11"].ArrayFormula = af;
}
```

Visual Basic

```
(FormulaActions.vb)
' Create an array formula that multiplies values contained in the cell range A2 through A11
' by the corresponding cells in the range B2 through B11,
' and displays the results in cells C2 through C11.
worksheet.Range("C2:C11").ArrayFormula = "=A2:A11*B2:B11"
' Create an array formula that multiplies values contained in the cell range C2 through C11 by 2
' and displays the results in cells D2 through D11.
worksheet.Range("D2:D11").ArrayFormula = "=C2:C11*2"
' Create an array formula that multiplies values contained in the cell range B2 through D11,
' adds the results, and displays the total sum in cell D12.
worksheet.Cells("D12").ArrayFormula = "=SUM(B2:B11*C2:C11*D2:D11)"
' Re-dimension an array formula range:
' delete the array formula and create a new range with the same formula.
If worksheet.Cells("C13").HasArrayFormula Then
    Dim af As String = worksheet.Cells("C13").ArrayFormula
    worksheet.Cells("C13").GetArrayFormulaRange().ArrayFormula = String.Empty
    worksheet.Range("C2:C11").ArrayFormula = af
End If
```

See Also
[Spreadsheet Formulas](#)
[Array Formulas](#)

Data Import and Export

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Import and Export Data](#)

This section contains the following examples.

- [How to: Import Data to a Worksheet](#)
- [How to: Export a Worksheet Range to a DataTable](#)

How to: Import Data to a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Import and Export Data](#) > [How to: Import Data to a Worksheet](#)

You can import data to worksheet cells from different data sources (for example, [arrays](#), [lists](#) and [data tables](#)). To import data into a worksheet, call the [WorksheetExtensions.Import](#) method and pass the following parameters:

- An object that specifies a source for importing data.
- Row and column indexes of the start cell in which you want to insert imported data into the worksheet.
- A Boolean value specifying whether to insert imported data vertically or horizontally. Use this parameter when importing data from a one-dimensional array or list.

Important

The [WorksheetExtensions](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the worksheet extensions. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

Import Data from Arrays

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

C#

(ImportActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
// Create an array containing string values.
string[] array = new string[] { "AAA", "BBB", "CCC", "DDD" };
// Import the array into the worksheet and insert it horizontally, starting with the B1 cell.
worksheet.Import(array, 0, 1, false);
// Create the two-dimensional array containing string values.
String[,] names = new String[2, 4]{
 {"Ann", "Edward", "Angela", "Alex"},
 {"Rachel", "Bruce", "Barbara", "George"}
};
// Import a two-dimensional array into the worksheet and insert it, starting with the B3 cell.
worksheet.Import(names, 2, 1);

Visual Basic

(ImportActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Create an array containing string values.
Dim array() As String = { "AAA", "BBB", "CCC", "DDD" }
' Import the array into the worksheet and insert it horizontally, starting with the B1 cell.
worksheet.Import(array, 0, 1, False)
' Create the two-dimensional array containing string values.
Dim names(,) As String = {
 {"Ann", "Edward", "Angela", "Alex"},
 {"Rachel", "Bruce", "Barbara", "George"}
}
' Import a two-dimensional array into the worksheet and insert it, starting with the B3 cell.
worksheet.Import(names, 2, 1)

Import Data from a List

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=E4339>

C#

```
(ImportActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
// Create the List object containing string values.
List<string> cities = new List<string>();
cities.Add("New York");
cities.Add("Rome");
cities.Add("Beijing");
cities.Add("Delhi");
// Import the list into the worksheet and insert it vertically, starting w
worksheet.Import(cities, 0, 0, true);
```

Import Data from a DataTable

Show Me

A complete sample project is available in the
DevExpress Code Examples database at
<http://www.devexpress.com/example=E4339>

C#


```
(ImportActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
// Create the "Employees" DataTable object with four columns.
DataTable table = new DataTable("Employees");
table.Columns.Add("FirstN", typeof(string));
table.Columns.Add("LastN", typeof(string));
table.Columns.Add("JobTitle", typeof(string));
table.Columns.Add("Age", typeof(Int32));
table.Rows.Add("Nancy", "Davolio", "recruiter", 32);
table.Rows.Add("Andrew", "Fuller", "engineer", 28);
// Import data from the data table into the worksheet and insert it, start
worksheet.Import(table, true, 0, 0);
// Color the table header.
for (int i = 1; i < 5; i++) {
    worksheet.Cells[10, i].FillColor = Color.LightGray;
}
```

Visual Basic

```
(ImportActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Create the "Employees" DataTable object with four columns.
Dim table As New DataTable("Employees")
table.Columns.Add("FirstN", GetType(String))
table.Columns.Add("LastN", GetType(String))
table.Columns.Add("JobTitle", GetType(String))
table.Columns.Add("Age", GetType(Int32))
table.Rows.Add("Nancy", "Davolio", "recruiter", 32)
table.Rows.Add("Andrew", "Fuller", "engineer", 28)
' Import data from the data table into the worksheet and insert it, starting at row 10.
worksheet.Import(table, True, 0, 0)
' Color the table header.
For i As Integer = 1 To 4
    worksheet.Cells(10, i).FillColor = Color.LightGray
Next i
```

The image below shows the result of executing the code above (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F
1	Import an array horizontally:	AAA	BBB	CCC	DDD	
2						
3	Import a two-dimensional array:	Ann	Edward	Angela	Alex	
4		Rachel	Bruce	Barbara	George	
5						
6	Import data from ArrayList vertically:	New York				
7		Rome				
8		Beijing				
9		Delhi				
10						
11	Import data from a DataTable:	FirstN	LastN	JobTitle	Age	
12		Nancy	Davolio	recruiter	32	
13		Andrew	Fuller	engineer	28	
14						
15						

 **Tip**

To import data into a worksheet, you can bind a read-only data source as described in the How to Bind a Spreadsheet to a List of Objects document and subsequently remove the binding using the WorksheetDataBindingCollection.Remove or WorksheetDataBindingCollection.Clear method.

How to: Export a Worksheet Range to a DataTable

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Import and Export Data](#) > [How to: Export a Worksheet Range to a DataTable](#)

You can export data from a worksheet cell range to a **DataTable**. In this case, worksheet columns are transformed into **DataTable** columns. Cell values are used to populate the DataTable, and you can specify conversion methods and control the conversion process for every cell (analyze data and modify values as required).

To export cell values to a data table, perform the following steps.

1. Create an empty **DataTable** that will be able to hold data of the worksheet range.

An empty DataTable, which will fit data contained in the specified worksheet range, can be created by using the **CreateDataTable** method of the Worksheet ([WorksheetExtensions.CreateDataTable](#)). The newly created DataTable contains the same number of columns as the worksheet range. Column data types are set automatically by analyzing the content of the first row in a range that contains data. Column names can be obtained from the first row of a range if the **rangeHasHeaders** method parameter is set to **true**.

Important

The [WorksheetExtensions](#) class is defined in the **DevExpress.Docs.v18.1.dll** assembly. Make sure you added a reference for this library to your project before using the worksheet extensions. Note that use of this library in production code requires a license to the DevExpress Office File API or DevExpress Universal Subscription. Refer to the [DevExpress Subscription](#) page for pricing information.

2. Create a DataTableExporter instance using the [WorksheetExtensions.CreateDataTableExporter](#) method.
3. Implement a custom converter for a specific DataTable column if required.

To accomplish this, create a class that implements the ICellValueToColumnTypeConverter interface. The ICellValueToColumnTypeConverter.Convert method should perform the required conversion. The **Convert** method is called for each cell that is exported to the specified column. The converter transforms **DateTime** values into strings in **MMMM-yyyy** format and displays the "N/A" text if a cell contains an error.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T482311>.

C#

```
(MyConverter.cs)
class MyConverter : ICellValueToColumnTypeConverter {
    public bool SkipErrorValues { get; set; }
    public CellValue EmptyCellValue { get; set; }
    public ConversionResult Convert(Cell readOnlyCell, CellValue cellValue, Type dataColumnType, out object result) {
        result = DBNull.Value;
        ConversionResult converted = ConversionResult.Success;
        if (cellValue.IsEmpty) {
            result = EmptyCellValue;
            return converted;
        }
        if (cellValue.IsError) {
            // You can return an error, subsequently the exporter throws an exception if the CellValueConverter.SkipErrorValues is false
            //return SkipErrorValues ? ConversionResult.Success : ConversionResult.Error;
            result = "N/A";
            return ConversionResult.Success;
        }
        result = String.Format("{0:MMMM-yyyy}", cellValue.DateTimeValue);
        return converted;
    }
}
```

Visual Basic

```

(MyConverter.vb)
Friend Class MyConverter
    Implements ICellValueToColumnTypeConverter
    Public Property SkipErrorValues() As Boolean
    Public Property EmptyCellValue() As CellValue Implements ICellValueToC
    Public Function Convert(ByVal readOnlyCell As Cell, ByVal cellValue As
        result = DBNull.Value
        Dim converted As ConversionResult = ConversionResult.Success
        If cellValue.IsEmpty Then
            result = EmptyCellValue
            Return converted
        End If
        If cellValue.IsError Then
            ' You can return an error, subsequently the exporter throws an
            'return SkipErrorValues ? ConversionResult.Success : Conversio
            result = "N/A"
            Return ConversionResult.Success
        End If
        result = String.Format("{0:MMMM-yyyy}", cellValue.DateTimeValue)
        Return converted
    End Function
End Class

```

Specify export options.

Create a new instance of the `DataTableExportOptions` class and specify the required options. Add an instance of the previously implemented custom converter to the collection of custom converters available through the `DataTableExportOptions.CustomConverters` property.

Call the [DataTableExporterExtensions.Export](#) method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T4823>
11.

C#

```

(Form1.cs)
Workbook wbook = new Workbook();
wbook.LoadDocument("TopTradingPartners.xlsx");
Worksheet worksheet = wbook.Worksheets[0];
Range range = worksheet.Tables[0].Range;
DataTable dataTable = worksheet.CreateDataTable(range, true);
// Change the data type of the "As Of" column to text.
dataTable.Columns["As Of"].DataType = System.Type.GetType("System.Stri
DataTableExporter exporter = worksheet.CreateDataTableExporter(range,
exporter.CellValueConversionError += exporter_CellValueConversionError
MyConverter myconverter = new MyConverter();
exporter.Options.CustomConverters.Add("As Of", myconverter);
// Set the export value for empty cell.
myconverter.EmptyCellValue = "N/A";
exporter.Options.ConvertEmptyCells = true;
exporter.Options.DefaultCellValueToColumnTypeConverter.SkipErrorValues
exporter.Export();
void exporter_CellValueConversionError(object sender, CellValueConversionE
MessageBox.Show("Error in cell " + e.Cell.GetReferenceA1());
e.DataTableValue = null;
e.Action = DataTableExporterAction.Continue;
}

```

Visual Basic

```

(Form1.vb)
Dim wbook As New Workbook()
wbook.LoadDocument("TopTradingPartners.xlsx")
Dim worksheet As Worksheet = wbook.Worksheets(0)
Dim range As Range = worksheet.Tables(0).Range
Dim dataTable As DataTable = worksheet.CreateDataTable(range, True)
' Change the data type of the "As Of" column to text.
dataTable.Columns("As Of").DataType = System.Type.GetType("System.Stri
Dim exporter As DataTableExporter = worksheet.CreateDataTableExporter(
AddHandler exporter.CellValueConversionError, AddressOf exporter_CellV
Dim myconverter As New MyConverter()
exporter.Options.CustomConverters.Add("As Of", myconverter)
' Set the export value for empty cell.
myconverter.EmptyCellValue = "N/A"
exporter.Options.ConvertEmptyCells = True
exporter.Options.DefaultCellValueToColumnTypeConverter.SkipErrorValues
exporter.Export()
Private Sub exporter_CellValueConversionError(ByVal sender As Object, ByVa
MessageBox.Show("Error in cell " & e.Cell.GetReferenceA1())
e.DataTableValue = Nothing
e.Action = DataTableExporterAction.Continue
End Sub

```

Data Binding

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Data Binding](#)

This section contains the following examples.

- [How to: Use Worksheet Table as a Data Source](#)
- [How to: Use Worksheet Data Bindings to Log and Process Data](#)

How to: Use Worksheet Table as a Data Source

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Data Binding](#) > [How to: Use Worksheet Table as a Data Source](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T518070>.

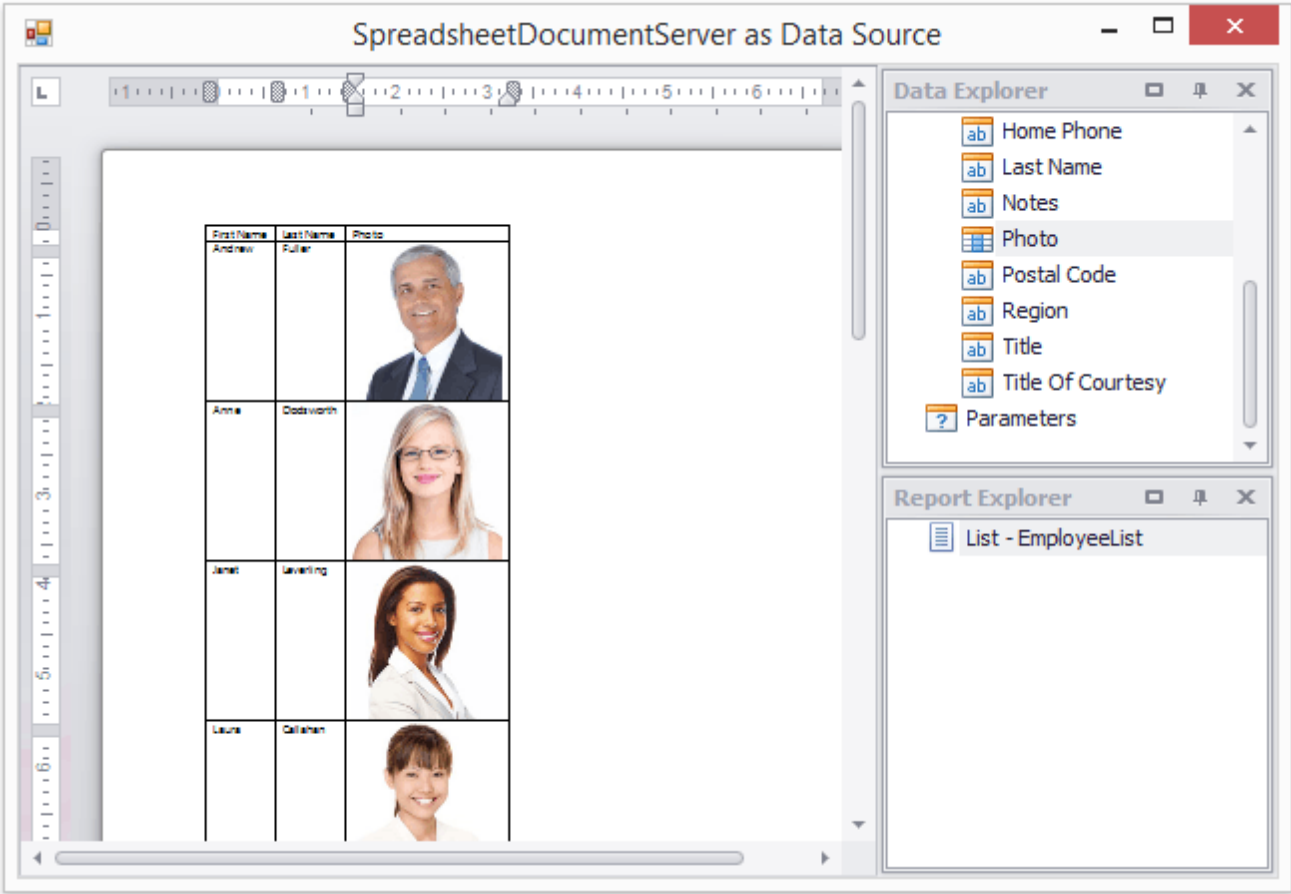
In this example, a worksheet containing a table with data is loaded in a [Workbook](#) instance. The Table.GetDataSource method returns a data source which is subsequently used to create a report in a SnapControl.

To modify worksheet data before they are exposed as data source fields, the application utilizes a custom **MyPictureProvider** converter which implements the IBindingRangeValueConverter interface. The MyPictureProvider converter finds a picture in a worksheet by its name and returns a picture bitmap instead of the name specified in a worksheet column.

The custom **MyColumnDetector** object which implements the IDataSourceColumnTypeDetector interface is used to specify column names and types.

The converter and column detector are specified using the DataSourceOptionsBase.CellValueConverter and RangeDataSourceOptions.DataSourceColumnTypeDetector properties of the RangeDataSourceOptions instance which is passed as a **Table.GetDataSource** method parameter.

Note that a Snap report template, which is required to visualize the data in the Snap™ application, is created in code at runtime. However, you can readily modify it using the Snap™ application UI.



C#

```
(Form1.cs)
DevExpress.Spreadsheet.Workbook wb =
    new DevExpress.Spreadsheet.Workbook();
wb.LoadDocument("Employees.xlsx");
DevExpress.Spreadsheet.RangeDataSourceOptions options =
    new DevExpress.Spreadsheet.RangeDataSourceOptions();
options.UseFirstRowAsHeader = true;
options.CellValueConverter = new MyPictureProvider(wb.Worksheets[0]);
options.DataSourceColumnTypeDetector = new MyColumnDetector();
string dsName = wb.Worksheets[0].Tables[0].Name;
object ds = wb.Worksheets[0].Tables[0].GetDataSource(options);
snapControl1.DataSources.Add(dsName, ds);
```

C#

```
(MyColumnDetector.cs)
class MyColumnDetector : IDataSourceColumnTypeDetector {
    public string GetColumnName(int index, int offset, Range range) {
        return range[-1, offset].DisplayText;
    }
    public Type GetColumnType(int index, int offset, Range range) {
        Type defaultType = typeof(string);
        if (offset == 13) return typeof(System.Drawing.Bitmap);
        CellValue value = range[0, offset].Value;
        if (value.IsText) return typeof(string);
        if (value.IsBoolean) return typeof(bool);
        if (value.IsDateTime) return typeof(DateTime);
        if (value.IsNumeric) return typeof(double);
        return defaultType;
    }
}
```

C#

```

(MyPictureProvider.cs)
public class MyPictureProvider : IBindingRangeValueConverter {
    Dictionary<string, Bitmap> pictures;
    public MyPictureProvider(Worksheet sheet) {
        pictures = GetPictures(sheet);
    }
    public object ConvertToObject(CellValue value, Type requiredType, int
        if (columnIndex == 13) {
            Bitmap pic;
            if (pictures.TryGetValue(value.TextValue, out pic))
                return pic;
        }
        return value;
    }
    public CellValue TryConvertFromObject(object value) {
        return CellValue.Empty;
    }
    public Dictionary<string, Bitmap> GetPictures(Worksheet sheet) {
        Dictionary<string, Bitmap> employeePictures = new Dictionary<string,
        foreach (Picture pic in sheet.Pictures) {
            employeePictures.Add(pic.Name, new Bitmap(new MemoryStream(pic
        }
        return employeePictures;
    }
}

```

Visual Basic

```

(Form1.vb)
Dim wb As New DevExpress.Spreadsheet.Workbook()
wb.LoadDocument("Employees.xlsx")
Dim options As New DevExpress.Spreadsheet.RangeDataSourceOptions()
options.UseFirstRowAsHeader = True
options.CellValueConverter = New MyPictureProvider(wb.Worksheets(0))
options.DataSourceColumnTypeDetector = New MyColumnDetector()
Dim dsName As String = wb.Worksheets(0).Tables(0).Name
Dim ds As Object = wb.Worksheets(0).Tables(0).GetDataSource(options)
snapControl1.DataSources.Add(dsName, ds)

```

Visual Basic

```
(MyColumnDetector.vb)
Friend Class MyColumnDetector
    Implements IDataSourceColumnTypeDetector
    Public Function GetColumnName(ByVal index As Integer, ByVal offset As
        Return range(-1, offset).DisplayText
    End Function
    Public Function GetColumnType(ByVal index As Integer, ByVal offset As
        Dim defaultType As Type = GetType(String)
        If offset = 13 Then
            Return GetType(System.Drawing.Bitmap)
        End If
        Dim value As CellValue = range(0, offset).Value
        If value.IsText Then
            Return GetType(String)
        End If
        If value.IsBoolean Then
            Return GetType(Boolean)
        End If
        If value.IsDateTime Then
            Return GetType(Date)
        End If
        If value.IsNumeric Then
            Return GetType(Double)
        End If
        Return defaultType
    End Function
End Class
```

Visual Basic

```
(MyPictureProvider.vb)
Public Class MyPictureProvider
    Implements IBindingRangeValueConverter
    Private pictures As Dictionary(Of String, Bitmap)
    Public Sub New(ByVal sheet As Worksheet)
        pictures = GetPictures(sheet)
    End Sub
    Public Function ConvertToObject(ByVal value As CellValue, ByVal columnIndex As Integer) As Object
        If columnIndex = 13 Then
            Dim pic As Bitmap = Nothing
            If pictures.TryGetValue(value.TextValue, pic) Then
                Return pic
            End If
        End If
        Return value
    End Function
    Public Function TryConvertFromObject(ByVal value As Object) As CellValue
        Return CellValue.Empty
    End Function
    Public Function GetPictures(ByVal sheet As Worksheet) As Dictionary(Of String, Bitmap)
        Dim employeePictures As Dictionary(Of String, Bitmap) = New Dictionary(Of String, Bitmap)
        For Each pic As Picture In sheet.Pictures
            employeePictures.Add(pic.Name, New Bitmap(New MemoryStream(pic.ImageData)))
        Next pic
        Return employeePictures
    End Function
End Class
```

How to: Use Worksheet Data Bindings to Log and Process Data

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Data Binding](#) > [How to: Use Worksheet Data Bindings to Log and Process Data](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T520862>.

This example illustrates the use of the [Workbook](#) instance to perform calculations on data bound to an external data source. The calculation result is located in the cell exposed as the data source for the GaugeControl control.

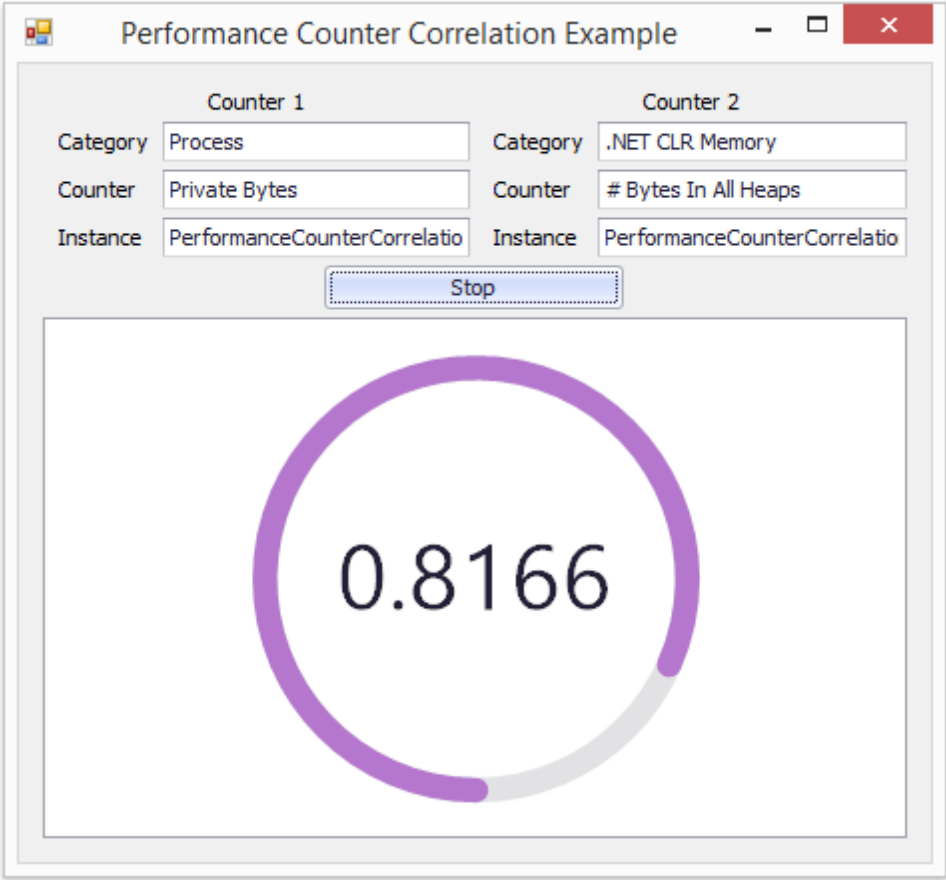
The application collects data from performance counters, logs them into a worksheet, calculates the correlation coefficient between data sets and gives a visual display of the coefficient value.

The source data are pairs of System.Diagnostics.PerformanceCounter object values. Performance samples (TwoCounterSample objects) are collected at the time interval specified by the timer. The samples are stored in the collection. A **WorksheetProcessor** object instance is created to process the collection of samples. A **WorksheetProcessor** includes a [Workbook](#) with a Worksheet containing a table bound to the performance data collection. The TableCollection.Add method is used to create a table bound to data.

The **CORREL** [statistical function](#) calculates the correlation coefficient. The cell containing the calculation result is exposed as the data source using the Range.GetDataSource method.

The worksheet data source providing a correlation coefficient value is bound to the ArcScaleComponent.DataBindings and the LabelComponent.DataBindings collections of the circular gauge control. The gauge control shows the correlation dynamically.

The application window is shown at the following picture.



C#

```
(WorksheetProcessor.cs)
public class WorksheetProcessor {
    Workbook wb;
    public WorksheetProcessor(object dataSource) {
        wb = new Workbook();
        wb.CreateNewDocument();
        Worksheet sheet = wb.Worksheets[0];
        Table sheetDataTable = sheet.Tables.Add(dataSource, sheet.Range["B2:C50"]);
        sheet.Cells["E2"].Formula = "IFERROR(CORREL(Table1[Column1], Table1[Column2]),0)";
    }
    public object GetDataSource() {
        return wb.Worksheets[0].Cells["E2"].GetDataSource();
    }
    public void SaveData(string fileNameBase) {
        string fileName = String.Format("{0:yyyy-MM-ddTHH-mm-ss}{1}.xlsx", DateTime.Now, fileNameBase);
        wb.SaveDocument(fileName, DocumentFormat.Xlsx);
    }
}
```

Visual Basic

```
(WorksheetProcessor.vb)
Public Class WorksheetProcessor
    Private wb As Workbook
    Public Sub New(ByVal dataSource As Object)
        wb = New Workbook()
        wb.CreateNewDocument()
        Dim sheet As Worksheet = wb.Worksheets(0)
        Dim sheetDataTable As Table = sheet.Tables.Add(dataSource, sheet.Range["B2:C50"])
        sheet.Cells("E2").Formula = "IFERROR(CORREL(Table1[Column1], Table1[Column2]),0)"
    End Sub
    Public Function GetDataSource() As Object
        Return wb.Worksheets(0).Cells("E2").GetDataSource()
    End Function
    Public Sub SaveData(ByVal fileNameBase As String)
        Dim fileName As String = String.Format("{0:yyyy-MM-ddTHH-mm-ss}{1}.xlsx", DateTime.Now, fileNameBase)
        wb.SaveDocument(fileName, DocumentFormat.Xlsx)
    End Sub
End Class
```

Sorting

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Sorting](#)

To sort data in a range in ascending or descending order, use the `Worksheet.Sort` method. Note that this method has several overloads with serious advantages. You can specify the column by which to sort a multi-column range, sort by multiple columns, or use a custom comparer.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

To sort data by multiple columns, follow the steps below.

1. Create the `SortField` object for each of the columns from which to sort. Specify the position of this column within the range using the `SortField.ColumnOffset` property and assign a comparer using the `SortField.Comparer` property. Instead of implementing your own comparer with the `System.Collections.Generic.IComparer`1[[DevExpress.Spreadsheet.CellValue]]` interface, you can use one of the built-in comparers which are responsible for sort operations performed by end-users. There are two built-in comparers - **Worksheet.Comparers.Ascending** is used for sorting in ascending order; **Worksheet.Comparers.Descending** is used for sorting in descending order.
2. Create a list containing sort fields.
3. Call the `Worksheet.Sort` method with two parameters: the first is the Range to be sorted, the other is the list of sort fields.

C#

(SortActions.cs)
Worksheet worksheet = workbook.Worksheets["SortSample"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create sorting fields.
List<SortField> fields = new List<SortField>();
// First sorting field. First column (offset = 0) will be sorted using ascending order.
SortField sortField1 = new SortField();
sortField1.ColumnOffset = 0;
sortField1.Comparer = worksheet.Comparers.Ascending;
fields.Add(sortField1);
// Second sorting field. Second column (offset = 1) will be sorted using ascending order.
SortField sortField2 = new SortField();
sortField2.ColumnOffset = 1;
sortField2.Comparer = worksheet.Comparers.Ascending;
fields.Add(sortField2);
// Sort the range by sorting fields.
Range range = worksheet.Range["A3:F22"];
worksheet.Sort(range, fields);

Visual Basic

```

(SortActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("SortSample")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create sorting fields.
Dim fields As New List(Of SortField)()
' First sorting field. First column (offset = 0) will be sorted using asce
Dim sortField1 As New SortField()
sortField1.ColumnOffset = 0
sortField1.Comparer = worksheet.Comparers.Ascending
fields.Add(sortField1)
' Second sorting field. Second column (offset = 1) will be sorted using as
Dim sortField2 As New SortField()
sortField2.ColumnOffset = 1
sortField2.Comparer = worksheet.Comparers.Ascending
fields.Add(sortField2)
' Sort the range by sorting fields.
Dim range As Range = worksheet.Range("A3:F22")
worksheet.Sort(range, fields)

```

The image below shows the result (the workbook is opened in Microsoft® Excel®). The cell range is sorted by the first and second columns in ascending order.

	A	B	C
1	G20 Countries Data as of 2010		
2	Continent	Country	Symbol
3	Asia	China	CHN
4	Asia	India	IND
5	Asia	Indonesia	IDN
6	Asia	Japan	JPN
7	Asia	Saudi Arabia	SAU
8	Asia	South Korea	KOR
9	Australia	Australia	AUS
10	Europe	France	FRA
11	Europe	Germany	DEU
12	Europe	Italy	ITA
13	Europe	Russian Federation	RUS
14	Europe	Spain	ESP

Mail Merge

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Mail Merge](#)

This section contains the following examples.

- [How to: Perform a Mail Merge](#)
- [How to: Validate the ObjectDataSource Contained in the Spreadsheet MailMerge Template](#)

How to: Perform a Mail Merge

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Mail Merge](#) > [How to: Perform a Mail Merge](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T515791>.

This example demonstrates how to use the [Spreadsheet Mail Merge](#) functionality to automatically generate a document based on data retrieved from an object data source. The [mail merge template](#) is created in code when the application starts. The data source is specified using the [Workbook.MailMergeDataSource](#) property. The [Workbook.GenerateMailMergeDocuments](#) method accomplishes mail merge and returns the resulting workbook. Since the mail merge mode is set to **Multiple Sheets**, all worksheets created by the `GenerateMailMergeDocuments` method are contained in a single workbook.

```
C#
(Program.cs)
Workbook workbook = new DevExpress.Spreadsheet.Workbook();
workbook.Unit = DevExpress.Office.DocumentUnit.Inch;
// Create a mail merge template.
Worksheet template = workbook.Worksheets[0];
template.Rows[1].RowHeight = 1.5;
template.Columns[1].ColumnWidth = 1.0;
template.Columns[1].Alignment.Vertical = SpreadsheetVerticalAlignment.Center;
template.Columns[2].ColumnWidth = 2.5;
template.Columns[2].Alignment.WrapText = true;
template.Cells["C2"].Formula = "FIELDPICTURE(\"Photo\", \"range\", C2, FALSE, 50)";
template.Cells["C3"].Formula = "=FIELD(\"FirstName\")&\" \"&FIELD(\"LastName\")";
template.Cells["B4"].Value = "Position:";
template.Cells["C4"].Formula = "FIELD(\"Title\")";
template.Cells["B5"].Value = "Birth Date:";
template.Cells["C5"].Formula = "FIELD(\"BirthDate\")";
template.Cells["C5"].NumberFormat = "M/d/yyyy";
template.Cells["B6"].Value = "Hire Date:";
template.Cells["C6"].Formula = "FIELD(\"HireDate\")";
template.Cells["C6"].NumberFormat = "dddd MMMM dd, yyyy";
template.Cells["B7"].Value = "Home Phone:";
template.Cells["C7"].Formula = "FIELD(\"HomePhone\")";
template.Cells["B8"].Value = "Address:";
template.Cells["C8"].Formula = "=FIELD(\"Address\")&\" \"&FIELD(\"City\")";
template.Cells["B9"].Value = "About:";
template.Cells["C9"].Formula = "FIELD(\"Notes\")";
// Set a detail range in the template.
Range detail = template.Range["C1:C9"];
detail.Name = "DETAILRANGE";
// Set a header range in the template.
Range header = template.Range["B1:B9"];
header.Name = "HEADERRANGE";
// Switch the mail merge mode to "Multiple Sheets".
workbook.DefinedNames.Add("MAILMERGEMODE", "=\"Worksheets\"");
// Switch the mail merge mode to "Multiple Documents".
//workbook.DefinedNames.GetDefinedName("MAILMERGEMODE").RefersTo = "\"Documents\"";
// Switch the mail merge mode to "Single Sheet".
//workbook.DefinedNames.GetDefinedName("MAILMERGEMODE").RefersTo = "\"OneWorksheet\"";
// Set vertical document orientation.
workbook.DefinedNames.Add("HORIZONTALMODE", "=TRUE");
// Perform mail merge.
workbook.MailMergeDataSource = EmployeeInfo.EmployeesInfo.GetData();
var result = workbook.GenerateMailMergeDocuments();
result[0].SaveDocument("result.xlsx");
System.Diagnostics.Process.Start("result.xlsx");
```

Visual Basic

```

(Program.vb)
Dim workbook As Workbook = New DevExpress.Spreadsheet.Workbook()
workbook.Unit = DevExpress.Office.DocumentUnit.Inch
' Create a mail merge template.
Dim template As Worksheet = workbook.Worksheets(0)
template.Rows(1).RowHeight = 1.5
template.Columns(1).ColumnWidth = 1.0
template.Columns(1).Alignment.Vertical = SpreadsheetVerticalAlignment.Cent
template.Columns(2).ColumnWidth = 2.5
template.Columns(2).Alignment.WrapText = True
template.Cells("C2").Formula = "FIELDPICTURE(""Photo"", ""range"", C2, FAI
template.Cells("C3").Formula = "=FIELD(""FirstName"")&"" ""&FIELD(""LastNa
template.Cells("B4").Value = "Position:"
template.Cells("C4").Formula = "FIELD(""Title"") "
template.Cells("B5").Value = "Birth Date:"
template.Cells("C5").Formula = "FIELD(""BirthDate"") "
template.Cells("C5").NumberFormat = "M/d/yyyy"
template.Cells("B6").Value = "Hire Date:"
template.Cells("C6").Formula = "FIELD(""HireDate"") "
template.Cells("C6").NumberFormat = "dddd MMMM dd, yyyy"
template.Cells("B7").Value = "Home Phone:"
template.Cells("C7").Formula = "FIELD(""HomePhone"") "
template.Cells("B8").Value = "Address:"
template.Cells("C8").Formula = "=FIELD(""Address"")&"" ""&FIELD(""City"") "
template.Cells("B9").Value = "About:"
template.Cells("C9").Formula = "FIELD(""Notes"") "
' Set a detail range in the template.
Dim detail As Range = template.Range("C1:C9")
detail.Name = "DETAILRANGE"
' Set a header range in the template.
Dim header As Range = template.Range("B1:B9")
header.Name = "HEADERRANGE"
' Switch the mail merge mode to "Multiple Sheets".
workbook.DefinedNames.Add("MAILMERGEMODE", "=""Worksheets"")
' Switch the mail merge mode to "Multiple Documents".
'workbook.DefinedNames.GetDefinedName("MAILMERGEMODE").RefersTo = "\"Docum
' Switch the mail merge mode to "Single Sheet".
'workbook.DefinedNames.GetDefinedName("MAILMERGEMODE").RefersTo = "\"OneWo
' Set vertical document orientation.
workbook.DefinedNames.Add("HORIZONTALMODE", "=TRUE")
' Perform mail merge.
workbook.MailMergeDataSource = EmployeeInfo.EmployeesInfo.GetData()
Dim result = workbook.GenerateMailMergeDocuments()
result(0).SaveDocument("result.xlsx")
System.Diagnostics.Process.Start("result.xlsx")

```

How to: Validate the ObjectDataSource Contained in the Spreadsheet MailMerge

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Mail Merge](#) > [How to: Validate the ObjectDataSource Contained in the Spreadsheet MailMerge Template](#)

Loading the mail merge templates with the ObjectDataSource data source may cause undesired behavior if the data source is contained in a compiled assembly. This example illustrates how to use a custom service that implements the IObjectDataSourceValidationService interface to validate an ObjectDataSource contained in the loaded mail merge template and prevent the data source from loading.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T515368>.

C#

```
(MyObjectDataSourceValidationService.cs)
public class MyObjectDataSourceValidationService : IObjectDataSourceValidationService {
    public void Validate(IEnumerable<ObjectDataSource> dataSources) {
        // Do nothing to allow loading.
        // Clear the DataSource and DataMember properties to prohibit loading.
        foreach (ObjectDataSource ds in dataSources) {
            if (ds.Name != "EmployeeDS") {
                ds.DataSource = null;
                ds.DataMember = null;
            }
        }
    }
}
```

C#

```
(Program.cs)
static void Main() {
    Workbook workbook = new DevExpress.Spreadsheet.Workbook();
    workbook.ReplaceService<IObjectDataSourceValidationService>(new MyObj
    workbook.LoadDocument("EmployeesMailMergeTemplate.xlsx");
    var result = workbook.GenerateMailMergeDocuments();
    result[0].SaveDocument("result.xlsx");
    System.Diagnostics.Process.Start("result.xlsx");
}
```

Visual Basic

```
(MyObjectDataSourceValidationService.vb)
Public Class MyObjectDataSourceValidationService
    Implements IObjectDataSourceValidationService
    Public Sub Validate(ByVal dataSources As IEnumerable(Of ObjectDataSou
        ' Do nothing to allow loading.
        ' Clear the DataSource and DataMember properties to prohibit loadi
        For Each ds As ObjectDataSource In dataSources
            If ds.Name <> "EmployeeDS" Then
                ds.DataSource = Nothing
                ds.DataMember = Nothing
            End If
        Next ds
    End Sub
End Class
```

Visual Basic

```
(Program.vb)
<STAThread> _
Shared Sub Main()
    Dim workbook As Workbook = New DevExpress.Spreadsheet.Workbook()
    workbook.ReplaceService(Of IObjectDataSourceValidationService) (New MyC
    workbook.LoadDocument("EmployeesMailMergeTemplate.xlsx")
    Dim result = workbook.GenerateMailMergeDocuments()
    result(0).SaveDocument("result.xlsx")
    System.Diagnostics.Process.Start("result.xlsx")
End Sub
```

Search

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Search](#)

To search for specific data in a range, worksheet or entire document, use the `Range.Search`, `Worksheet.Search` or `Workbook.Search` methods, respectively. To set options affecting search in a document, create an instance of the `SearchOptions` class and pass it as a parameter to the **Search** method. You can set the following advanced options.

- To specify the direction of the search (whether to perform a search by rows or by columns), use the `SearchOptions.SearchBy` property.
- To specify what to examine in each cell when searching (cell values only or cell values with formulas), use the `SearchOptions.SearchIn` property.
- To perform a case-sensitive search, set the `SearchOptions.MatchCase` property to **true**.
- To search for an exact match of characters specified by the search term, set the `SearchOptions.MatchEntireCellContents` property to **true**.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(SearchActions.cs)
workbook.Calculate();
Worksheet worksheet = workbook.Worksheets["ExpenseReport"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Specify the search term.
string searchString = DateTime.Today.ToString("d");
// Specify search options.
SearchOptions options = new SearchOptions();
options.SearchBy = SearchBy.Columns;
options.SearchIn = SearchIn.Values;
options.MatchEntireCellContents = true;
// Find all cells containing today's date and paint them light-green.
IEnumerable<Cell> searchResult = worksheet.Search(searchString, options);
foreach (Cell cell in searchResult)
    cell.Fill.BackgroundColor = Color.LightGreen;
```

Visual Basic

```
(SearchActions.vb)
workbook.Calculate()
Dim worksheet As Worksheet = workbook.Worksheets("ExpenseReport")
workbook.Worksheets.ActiveWorksheet = worksheet
' Specify the search term.
Dim searchString As String = Date.Today.ToString("d")
' Specify search options.
Dim options As New SearchOptions()
options.SearchBy = SearchBy.Columns
options.SearchIn = SearchIn.Values
options.MatchEntireCellContents = True
' Find all cells containing today's date and paint them light-green.
Dim searchResult As IEnumerable(Of Cell) = worksheet.Search(searchString,
For Each cell As Cell In searchResult
    cell.Fill.BackgroundColor = Color.LightGreen
Next cell
```

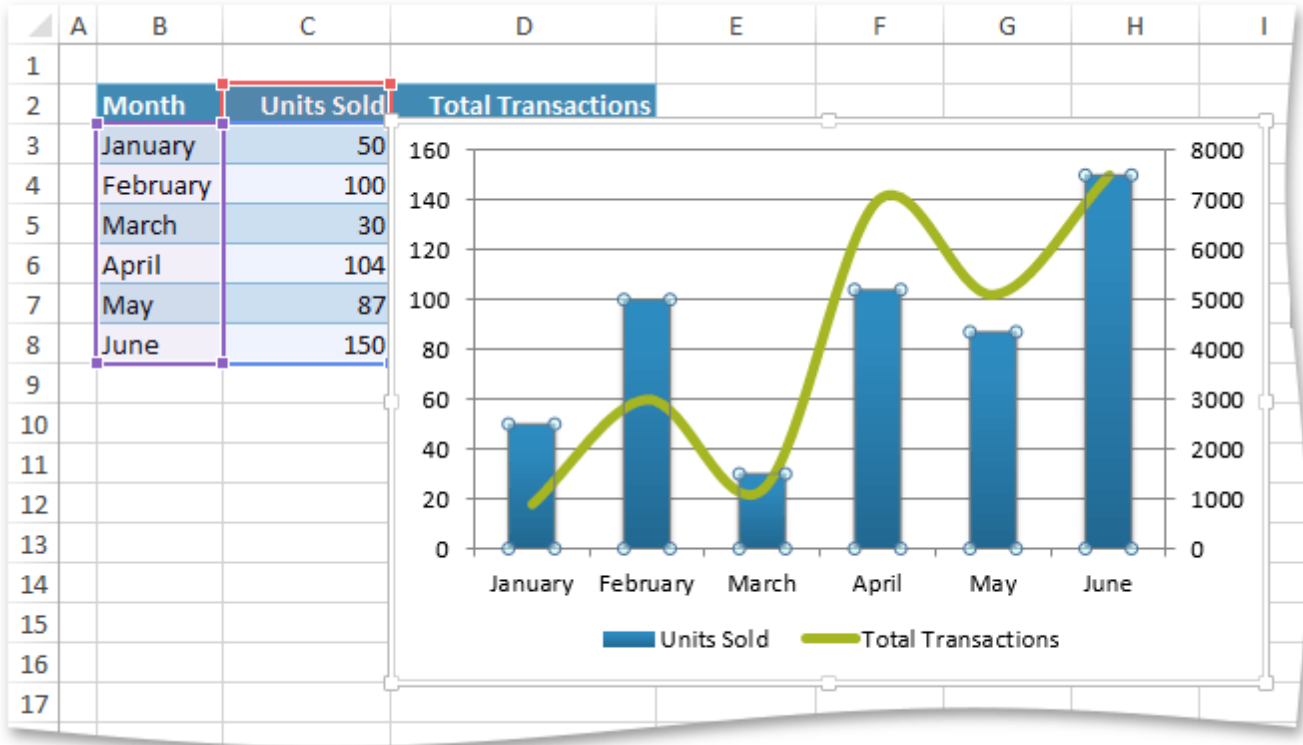
The image below shows the result of executing the code (the workbook is opened in Microsoft® Excel®). Today's date is located in the expense report and highlighted in light-green.

	A	B	C	D	E	F	G
3		Date	Account	Description		Hotel	
4		3/6/2015	199123	Business trip		\$ 349.00	
5		3/7/2015	423509	Holiday		\$ -	
6		3/8/2015	543288	Holiday		\$ -	
7		3/9/2015	457382	Business trip		\$ 205.00	
8		3/10/2015	584839	Business trip		\$ 205.00	
9		3/11/2015	483922	Business trip		\$ 205.00	
10		3/12/2015	890763	Business trip		\$ 205.00	
11						\$ 1,169.00	

Charts

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Charts](#)

This section contains examples that demonstrate how to create and modify charts to visualize large amounts of data in a worksheet. You can create a 2-D or 3-D chart, and fully customize the appearance of any chart element.



- [How to: Create a Basic Chart](#)
- [How to: Format Chart Elements](#)
- [How to: Display the Chart Title](#)
- [How to: Display and Format Data Label](#)
- [How to: Show or Hide the Chart Legend](#)
- [How to: Change the Display of Chart Axes](#)
- [How to: Protect a Chart](#)

How to: Create a Basic Chart

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Charts](#) > [How to: Create a Basic Chart](#)

These examples demonstrate how to create a basic chart and specify its settings.

Select the action you wish to perform.

- [Add and Position a Chart](#)
- [Add or Remove Data Series](#)
- [Change Chart Data References](#)
- [Change the Order of Data Series](#)
- [Change the Chart Type](#)
- [Change the Chart Type of a Series \(Combination Chart\)](#)

Add and Position a Chart

All charts embedded in a worksheet are stored in the chart collection accessible using the `Worksheet.Charts` property. To create a chart, add it to the chart collection by utilizing the `ChartCollection.Add` method and pass the following parameters: the required chart type (`ChartType`) and the `Range` object containing data for the chart (this parameter is optional and can be omitted, so you can manually [add data series](#) to your chart).

To position the created chart in a worksheet, use one of the following approaches.

- Anchor the chart to cells by specifying the chart's `FloatingObject.TopLeftCell` and `FloatingObject.BottomRightCell` properties. The chart will move and resize with the underlying cells (`Placement.MoveAndSize`).
- Set the `FloatingObject.Top` and `FloatingObject.Left` offsets in the worksheet for the chart's top left corner. The chart will be a free floating shape (`Placement.FreeFloating`).

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a 3-D pie chart using the `ChartCollection.Add` method overload that allows you to specify a `Range` containing chart data.

C#

```
(CreationAndDataActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a pie chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.Pie3D, worksheet["B2:C7"]);
chart.TopLeftCell = worksheet.Cells["E2"];
chart.BottomRightCell = worksheet.Cells["K15"];
// Set the chart style.
chart.Style = ChartStyle.ColorGradient;
```

Visual Basic

```
(CreationAndDataActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a pie chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.Pie3D, worksheet("B2:C7"))
chart.TopLeftCell = worksheet.Cells("E2")
chart.BottomRightCell = worksheet.Cells("K15")
' Set the chart style.
chart.Style = ChartStyle.ColorGradient
```

Add or Remove Data Series

If you did not specify the range containing chart data in the `ChartCollection.Add` method, you can define it later by using one of the following approaches.

- Pass the required data range to the `ChartObject.SelectData` method. You can also specify the direction in which data should be plotted along the axes. For example, if you wish to display columns of data along the [category axis](#), use the `ChartDataDirection.Column` value as the *direction* parameter.
- Utilize the `SeriesCollection.Add` method to plot a data series on your chart. Using this method, you can fully control the number of series on your chart and explicitly specify data that should be plotted along the [category](#) and [value axes](#). Moreover, this method is extremely useful if you wish to plot data contained in noncontiguous ranges. For example, you can plot a data series with arguments from Column **B** and values from the Column **D**.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#
<pre>(CreationAndDataActions.cs) Worksheet worksheet = workbook.Worksheets["chartTask3"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Create a chart and specify its location. Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered); chart.TopLeftCell = worksheet.Cells["H2"]; chart.BottomRightCell = worksheet.Cells["N14"]; // Add chart series using worksheet ranges as the data sources. chart.Series.Add(worksheet["D2"], worksheet["B3:B6"], worksheet["D3:D6"]); chart.Series.Add(worksheet["F2"], worksheet["B3:B6"], worksheet["F3:F6"]);</pre>

Visual Basic
<pre>(CreationAndDataActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask3") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered) chart.TopLeftCell = worksheet.Cells("H2") chart.BottomRightCell = worksheet.Cells("N14") ' Add chart series using worksheet ranges as the data sources. chart.Series.Add(worksheet("D2"), worksheet("B3:B6"), worksheet("D3:D6")) chart.Series.Add(worksheet("F2"), worksheet("B3:B6"), worksheet("F3:F6"))</pre>

To remove an individual series from the chart, use the `SeriesCollection.Remove` or `SeriesCollection.RemoveAt` method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to remove the second series from the chart.

C#

```
(SeriesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Remove the series.
chart.Series.RemoveAt(1);
```

Visual Basic
<pre>(SeriesActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask3") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works chart.TopLeftCell = worksheet.Cells("H2") chart.BottomRightCell = worksheet.Cells("N14") ' Remove the series. chart.Series.RemoveAt(1)</pre>

Change Chart Data References

After you create a chart, you can change the cell range from which data for the series arguments and values is retrieved. To do this, perform the steps below.

- 1. Use the ChartData.FromRange method to create the ChartData object containing new data for the series.
- 2. Assign the created object to the Series.Arguments or Series.Values property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to add data series to a chart and change the cell range containing data for a specific series.

The code uses the SeriesCollection.Add method to create a series and add it to the ChartObject.Series collection. To specify series values contained in the worksheet range, create a ChartData object using the ChartData.FromRange method and assign it to the Series.Values property.

The ChartText.SetReference method is used to link the Series.SeriesName to the content of the worksheet cell.

C#

```
(CreationAndDataActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered);
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Add series using a worksheet range as the data source.
chart.Series.Add(worksheet["D2"], worksheet["B3:B6"], worksheet["D3:D6"]);
chart.Series.Add(worksheet["F2"], worksheet["B3:B6"], worksheet["F3:F6"]);
// Change the data range for the series values.
chart.Series[1].Values = ChartData.FromRange(worksheet["E3:E6"]);
// Specify the cell that is the source for the series name.
chart.Series[1].SeriesName.SetReference(worksheet["E2"]);
```

Visual Basic	
<pre>(CreationAndDataActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask3") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered) chart.TopLeftCell = worksheet.Cells("H2") chart.BottomRightCell = worksheet.Cells("N14") ' Add series using a worksheet range as the data source. chart.Series.Add(worksheet("D2"), worksheet("B3:B6"), worksheet("D3:D6")) chart.Series.Add(worksheet("F2"), worksheet("B3:B6"), worksheet("F3:F6")) ' Change the data range for the series values. chart.Series(1).Values = ChartData.FromRange(worksheet("E3:E6")) ' Specify the cell that is the source for the series name. chart.Series(1).SeriesName.SetReference(worksheet("E2"))</pre>	

Change the Order of Data Series

By default, if your chart has multiple data series, they are plotted on a chart in the order they appear on a worksheet (from left to right if data is arranged in columns, or from top to bottom if data is arranged in rows). However, you can change this default order. Utilize the Series.BringForward or Series.SendBackward method to move the specified series up or down in the plotting order, one position at a time. To place the series before or behind all other series on a chart, use the Series.BringToFront or Series.SendToBack method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#	
----	--

```
(SeriesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Change the series order.
chart.Series[1].BringForward();
```

Visual Basic

```
(SeriesActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Change the series order.
chart.Series(1).BringForward()
```

Change the Chart Type

After you created a chart, you may find that the chart you selected is not suitable and another chart type may better represent your data. In this case, there is no need to recreate the chart. Simply change the type of the entire chart using the `ChartObject.ChangeType` method. This method substitutes the existing chart with the chart you specified. If the operation cannot be completed, an exception is fired.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a chart of the `ChartType.PieExploded` type. After that, the code attempts to change the chart type to `ChartType.LineMarker` using the `ChartObject.ChangeType` method. If you try to change the chart type to `ChartType.StockHighLowClose` (this line is commented), an exception will be thrown, because the data range is insufficient for this chart type, and the chart will be changed to `ChartType.ColumnClustered`.

C#

```
(Charts.cs)
Worksheet worksheet = workbook.Worksheets["chartTask1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
ChartType type1 = ChartType.LineMarker;
// If a new chart type cannot be created with existing data, an exception
//ChartType type1 = ChartType.StockHighLowClose;
ChartType type2 = ChartType.ColumnClustered;
// Create a chart and specify its location
Chart chart = worksheet.Charts.Add(ChartType.PieExploded, worksheet["B2:C7"]);
// Change the chart type.
try
{
    chart.ChangeType(type1);
}
catch (Exception e)
{
    MessageBox.Show(e.Message, "Incompatible chart type");
    chart.ChangeType(type2);
}
```

Visual Basic
<pre>(Charts.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask1") workbook.Worksheets.ActiveWorksheet = worksheet Dim type1 As ChartType = ChartType.LineMarker ' If a new chart type cannot be created with existing data, an exception is 'ChartType type1 = ChartType.StockHighLowClose; Dim type2 As ChartType = ChartType.ColumnClustered ' Create a chart and specify its location Dim chart As Chart = worksheet.Charts.Add(ChartType.PieExploded, worksheet["B2:C7"]); ' Change the chart type. Try chart.ChangeType(type1) Catch e As Exception MessageBox.Show(e.Message, "Incompatible chart type") chart.ChangeType(type2) End Try</pre>

Change the Chart Type of a Series (Combination Chart)

Besides [changing a type of the entire chart](#), you can also select a different chart type for an individual data series. This will automatically turn the chart into a combination chart. A combination chart is a complex chart that consists of two or more **ChartView** objects that consolidate series of the same chart type. To change the type of a single data series, call the **Series.ChangeType** method. If the series type you specified differs from the types of the existing **ChartView** objects (**ChartView.ViewType**), a new **ChartView** containing your series will be created, otherwise, the series will be added to the existing **ChartView** of the same type.

Important
Not all chart types can be combined (for example, it is impossible to combine the 2-D and 3-D chart types). If you select a type that does not logically connect with the

existing chart type, the resulting chart will contain only one ChartView of the most recently added type.

Compatible chart types are listed below.

- ChartType.Area
- ChartType.AreaFullStacked
- ChartType.AreaStacked
- ChartType.BarClustered
- ChartType.BarFullStacked
- ChartType.BarStacked
- ChartType.ColumnClustered
- ChartType.ColumnFullStacked
- ChartType.ColumnStacked
- ChartType.Line
- ChartType.LineFullStacked
- ChartType.LineFullStackedMarker
- ChartType.LineMarker
- ChartType.LineStacked
- ChartType.LineStackedMarker

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#	
<pre>(SeriesActions.cs) Worksheet worksheet = workbook.Worksheets["chartTask5"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Create a chart and specify its location. Chart chart = worksheet.Charts.Add(ChartType.LineMarker, worksheet["B2:D8"] chart.TopLeftCell = worksheet.Cells["F2"]; chart.BottomRightCell = worksheet.Cells["L15"]; // Change the type of the second series. chart.Series[1].ChangeType(ChartType.ColumnClustered); // Use the secondary axis. chart.Series[1].AxisGroup = AxisGroup.Secondary; // Specify the position of the legend. chart.Legend.Position = LegendPosition.Top;</pre>	
Visual Basic	

```
(SeriesActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask5")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.LineMarker, worksheet (
chart.TopLeftCell = worksheet.Cells("F2")
chart.BottomRightCell = worksheet.Cells("L15")
' Change the type of the second series.
chart.Series(1).ChangeType(ChartType.ColumnClustered)
' Use the secondary axis.
chart.Series(1).AxisGroup = AxisGroup.Secondary
' Specify the position of the legend.
chart.Legend.Position = LegendPosition.Top
```

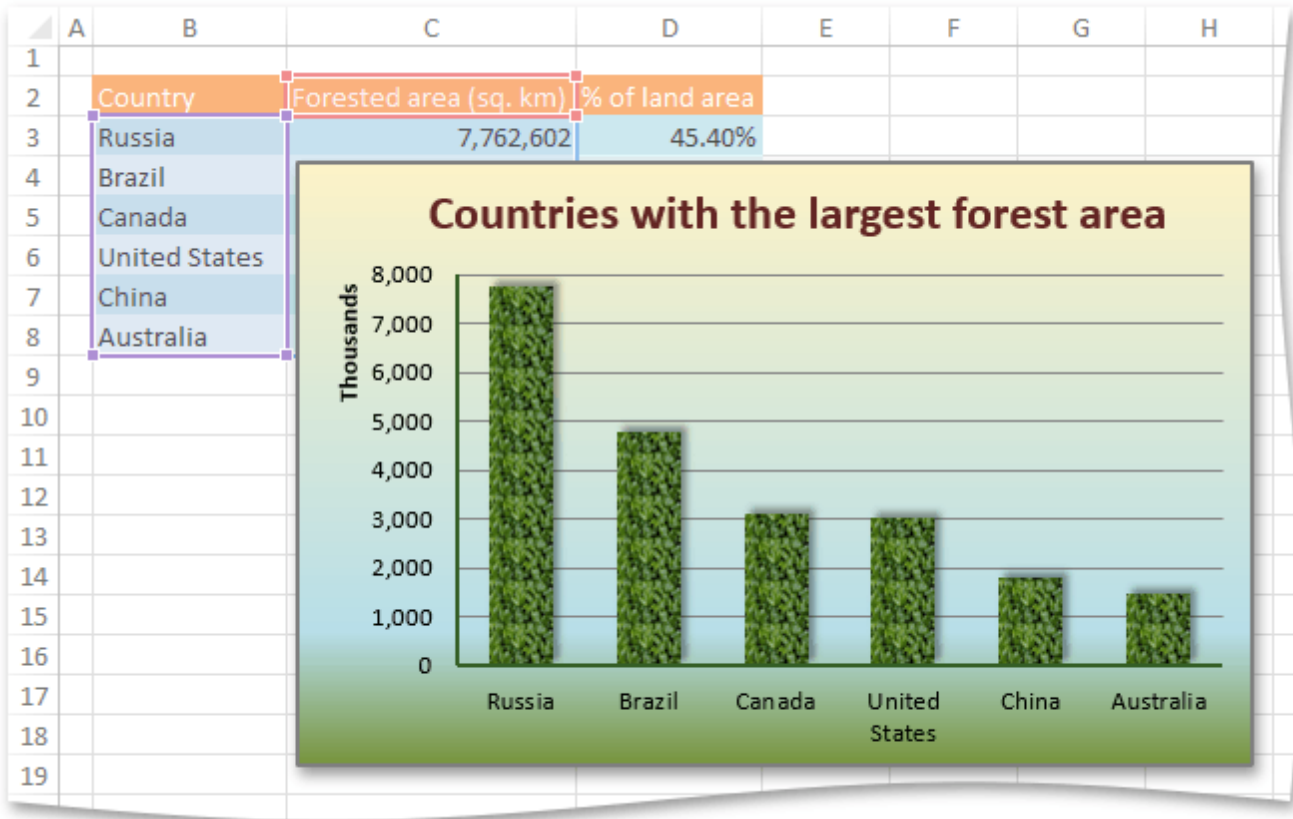
See Also

[How to: Format Chart Elements](#)

How to: Format Chart Elements

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Charts](#) > [How to: Format Chart Elements](#)

This example demonstrates how to enhance the appearance of an existing chart. For an example on how to create a basic chart in code, refer to the [How to: Create a Basic Chart](#) article.



Select the action you wish to perform.

- [Apply the Built-in Chart Style](#)
- [Format an Individual Chart Element](#)
- [Specify Chart View Options](#)

Apply the Built-in Chart Style

Chart styles allow you to quickly change chart appearance. The chart style changes the chart's background fill, specifies the color of the data series, and applies different shape effects and outlines to the chart. To apply one of the predefined styles to the chart, utilize the `ChartObject.Style` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a chart and apply one of the predefined styles to it using the `ChartObject.Style` property.

C#

```
(StyleActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B2:D4"]);
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Set the chart style.
chart.Style = ChartStyle.Accent1Dark;
```

Visual Basic

```
(StyleActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet("B2:D4"))
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Set the chart style.
chart.Style = ChartStyle.Accent1Dark
```

Format an Individual Chart Element

The chart style allows you to apply a predefined set of format options. However, you can fine-tune these settings and specify custom formatting for individual chart elements by giving them a different color or outline. A full set of format characteristics available for a chart is provided by the `ShapeFormat` object. All objects that represent chart elements (`Chart`, `PlotArea`, `Series`, `Axis`, `DataLabel`, `Legend`, `ChartTitle`, etc.) inherit the `ShapeFormat` interface, so you can use its formatting properties to set a color and border of the required chart element.

• Fill a chart element

To fill a chart element, use the `ShapeFormat.Fill` property. This property returns the `ShapeFill` object that contains the shape fill characteristics. You can fill a chart element with a solid color by using the `ShapeOutlineFill.SetSolidFill` method, make an element transparent by utilizing the `ShapeOutlineFill.SetNoFill` method, apply the gradient effect by calling the `ShapeOutlineFill.SetGradientFill` method, or specify a pattern or picture fill by using the `ShapeFill.SetPatternFill` or `ShapeFill.SetPictureFill` method.

• Specify the outline of a chart element

To set a border of a chart element, use the `ShapeFormat.Outline` property. This property provides access to the `ShapeOutline` object that contains options used to draw the border or line on a chart. Use the `ShapeOutlineFill.SetSolidFill` method to specify border or line color. To remove the outline, call the `ShapeOutlineFill.SetNoFill` method. You can also specify the line width (`ShapeOutline.Width`), adjust the appearance of the line end (`ShapeOutline.CapType`), set a dash style for a chart line (`ShapeOutline.Dashing`), specify a join type (`ShapeOutline.JoinType`) and compound style (`ShapeOutline.CompoundType`) for the chart elements.

• Format a text of a chart element

To change font attributes for all chart elements at once, use the `ChartObject.Font` property. To format a text contained in a specific chart element, use the element's `ShapeTextFormat.Font` property. These properties provide access to the `ShapeTextFont` object that contains a set of properties you can use to change font characteristics for the chart and axis titles, tick-mark labels and data labels. To change the font attributes for legend entries, use the `Legend.Font` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#	
<pre>(ViewOptionsActions1.cs) Worksheet worksheet = workbook.Worksheets["chartTask7"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Create a chart and specify its location. Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B chart.TopLeftCell = worksheet.Cells["F2"]; chart.BottomRightCell = worksheet.Cells["N17"]; // Add and format the chart title. chart.Title.SetValue("?ountries with the largest forest area"); chart.Title.Font.Color = Color.FromArgb(0x34, 0x5E, 0x25); // Set no fill for the plot area. chart.PlotArea.Fill.SetNoFill(); // Apply the gradient fill to the chart area. chart.Fill.SetGradientFill(ShapeGradientType.Linear, Color.FromArgb(0xFD, ShapeGradientFill gradientFill = chart.Fill.GradientFill; gradientFill.Stops.Add(0.78f, Color.FromArgb(0xB7, 0xDE, 0xE8)); gradientFill.Angle = 90; // Set the picture fill for the data series. chart.Series[0].Fill.SetPictureFill("Pictures\\PictureFill.png"); // Customize the axis appearance. AxisCollection axisCollection = chart.PrimaryAxes; foreach (Axis axis in axisCollection) { axis.MajorTickMarks = AxisTickMarks.None; axis.Outline.SetSolidFill(Color.FromArgb(0x34, 0x5E, 0x25)); axis.Outline.Width = 1.25; } // Change the scale of the value axis. Axis valueAxis = axisCollection[1]; valueAxis.Scaling.AutoMax = false; valueAxis.Scaling.Max = 8000000; valueAxis.Scaling.AutoMin = false; valueAxis.Scaling.Min = 0; // Specify display units for the value axis. valueAxis.DisplayUnits.UnitType = DisplayUnitType.Thousands; valueAxis.DisplayUnits.ShowLabel = true; // Hide the legend. chart.Legend.Visible = false;</pre>	
Visual Basic	

```

(ViewOptionsActions1.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask7")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works
chart.TopLeftCell = worksheet.Cells("F2")
chart.BottomRightCell = worksheet.Cells("N17")
' Add and format the chart title.
chart.Title.SetValue("?ountries with the largest forest area")
chart.Title.Font.Color = Color.FromArgb(&H34, &H5E, &H25)
' Set no fill for the plot area.
chart.PlotArea.Fill.SetNoFill()
' Apply the gradient fill to the chart area.
chart.Fill.SetGradientFill(ShapeGradientType.Linear, Color.FromArgb(&HFD,
Dim gradientFill As ShapeGradientFill = chart.Fill.GradientFill
gradientFill.Stops.Add(0.78F, Color.FromArgb(&HB7, &HDE, &HE8))
gradientFill.Angle = 90
' Set the picture fill for the data series.
chart.Series(0).Fill.SetPictureFill("Pictures\PictureFill.png")
' Customize the axis appearance.
Dim axisCollection As AxisCollection = chart.PrimaryAxes
For Each axis As Axis In axisCollection
    axis.MajorTickMarks = AxisTickMarks.None
    axis.Outline.SetSolidFill(Color.FromArgb(&H34, &H5E, &H25))
    axis.Outline.Width = 1.25
Next axis
' Change the scale of the value axis.
Dim valueAxis As Axis = axisCollection(1)
valueAxis.Scoring.AutoMax = False
valueAxis.Scoring.Max = 8000000
valueAxis.Scoring.AutoMin = False
valueAxis.Scoring.Min = 0
' Specify display units for the value axis.
valueAxis.DisplayUnits.UnitType = DisplayUnitType.Thousands
valueAxis.DisplayUnits.ShowLabel = True
' Hide the legend.
chart.Legend.Visible = False

```

Specify the Chart View Options

There are specific settings for different chart types that affect chart appearance. Use the properties of the `ChartView` and `View3DOptions` objects to specify chart options for 2-D and 3-D charts. For example, you can adjust the distance between column clusters in the column chart by using the `ChartView.GapWidth` property, or apply a different color to each data marker in the series by setting the `ChartView.VaryColors` property to **true**.

🔗 Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a clustered column chart and automatically apply a different color to each data marker representing a data point on the chart. To vary colors of the same-series data markers point by point, set the `ChartView.VaryColors` property to **true**.

C#	
<pre>(ViewOptionsActions.cs) Worksheet worksheet = workbook.Worksheets["chartTask5"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Create a chart and specify its location. Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B chart.TopLeftCell = worksheet.Cells["F2"]; chart.BottomRightCell = worksheet.Cells["L15"]; // Specify that each data point in the series has a different color. chart.Views[0].VaryColors = true; // Hide the legend. chart.Legend.Visible = false;</pre>	

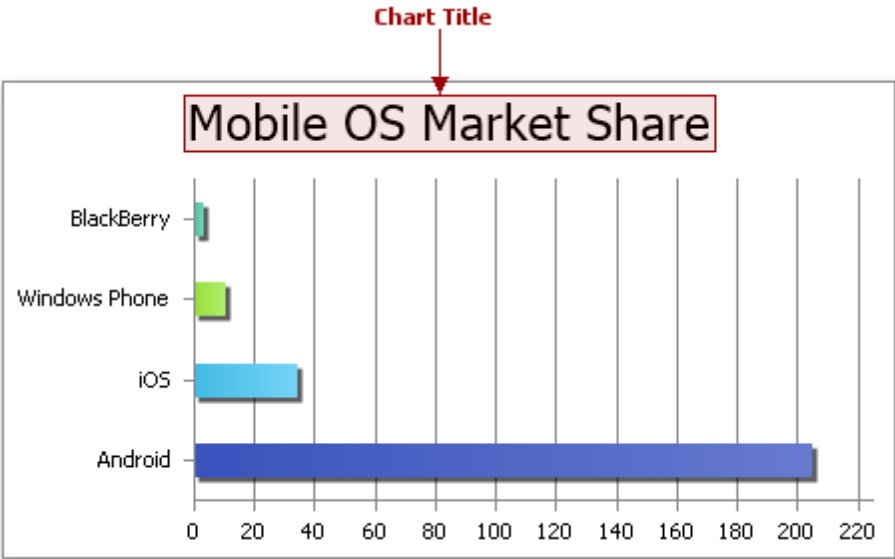
Visual Basic	
<pre>(ViewOptionsActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask5") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works chart.TopLeftCell = worksheet.Cells("F2") chart.BottomRightCell = worksheet.Cells("L15") ' Specify that each data point in the series has a different color. chart.Views(0).VaryColors = True ' Hide the legend. chart.Legend.Visible = False</pre>	

See Also
[How to: Create a Basic Chart](#)

How to: Display the Chart Title

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Charts](#) > [How to: Display the Chart Title](#)

After you [create a chart](#), you can display its title (a text header that explains the purpose of a chart). By default, the chart title is displayed at the top of the chart.



To add the chart title, utilize the `ChartObject.Title` property, which provides access to the `ChartTitle` object. This object defines the main chart title and inherits basic title settings from the `ChartTitleOptions` and `ChartText` interfaces, which contain members used to display the title and specify its contents. Set the `ChartTitleOptions.Visible` property to **true** to add the title, and then do one of the following.

- To specify the title text manually, use the `ChartText.SetValue` method.
- To extract the text for the title from the cell range, use the `ChartText.SetReference` method.

You can also format the chart title by changing its font, color, size or location. For information on how to change the format options of chart elements, refer to the [How to: Format Chart Elements](#) example.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#

```
(TitlesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask2"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.BarClustered, worksheet["B4:C7"]);
chart.TopLeftCell = worksheet.Cells["E3"];
chart.BottomRightCell = worksheet.Cells["K14"];
// Display the chart title and specify the title text.
chart.Title.Visible = true;
chart.Title.SetValue("Market share Q3'13");
// Hide the chart legend.
chart.Legend.Visible = false;
// Specify that each data point in the series has a different color.
chart.Views[0].VaryColors = true;
```

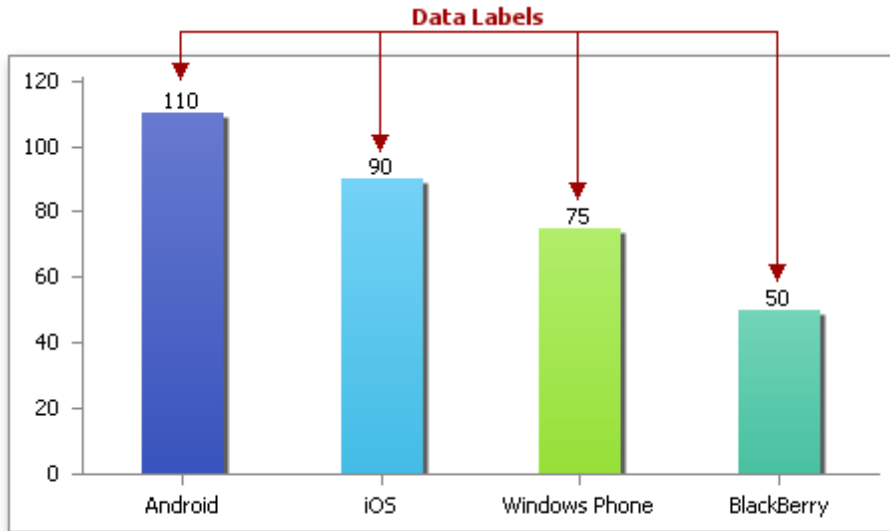
Visual Basic

```
(TitlesActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask2")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.BarClustered, worksheet)
chart.TopLeftCell = worksheet.Cells("E3")
chart.BottomRightCell = worksheet.Cells("K14")
' Display the chart title and specify the title text.
chart.Title.Visible = True
chart.Title.SetValue("Market share Q3'13")
' Hide the chart legend.
chart.Legend.Visible = False
' Specify that each data point in the series has a different color.
chart.Views(0).VaryColors = True
```

How to: Display and Format Data Label

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Charts](#) > [How to: Display and Format Data Label](#)

After you [create a chart](#), you can add a data label to each data point in the chart to identify its actual value. By default, data labels are linked to data that the chart uses. When data changes, information in the data labels is updated automatically. If required, you can also display custom information in a label.



Select the action you wish to perform.

- [Add Data Labels to the Chart](#)
- [Specify the Position of Data Labels](#)
- [Apply Number Format to Data Labels](#)
- [Create a Custom Label Entry](#)

Add Data Labels to the Chart

Basic settings that specify the contents, position and appearance of data labels in the chart are defined by the `DataLabelOptions` object, accessed by the `ChartView.DataLabels` property. Use the object's properties to display data labels for all series in the current chart view and specify what information should appear in each label. The following options are available:

- **Value.** Value labels identify the underlying value of each data point. To display value labels, set the `DataLabelBase.ShowValue` property of the `DataLabelOptions` object to **true**.
- **Series name.** Series labels identify data series to which the data points in the chart belong. Most series include multiple data points, so the same name will be repeated for all data points in the series, which is probably overkill. However, if you wish to display series name labels, utilize the `DataLabelBase.ShowSeriesName` property of the `DataLabelOptions` object.
- **Category name.** Category labels repeat information of the category axis labels. This option can be useful if you wish to display the category names for charts that do not have a category axis, like a *pie* or *doughnut* chart. To display the category labels, use the `DataLabelBase.ShowCategoryName` property.
- **Percentage.** Percentage labels are available for the *pie* and *doughnut* chart types only. They display a percentage calculated by using the basic formula that divides the data point value by the total of all values in the series. To add the percentage labels, utilize the `DataLabelBase.ShowPercent` property.
- **Bubble size.** Bubble size labels display the bubble size value in a *bubble* chart. Since the bubble size is a third dimension that does not correspond to any axis and is shown as a filled shape of a particular size, you can identify its value by displaying the bubble size labels at each data point. To do this, use the `DataLabelBase.ShowBubbleSize` property.
- **Legend key.** Legend key adds a sample of the color and fill pattern used to draw the corresponding series in a chart. To display the legend key next to each data label, utilize `DataLabelBase.ShowLegendKey` property.

If you use a combination of data label entries, you can specify a character to separate different items in a label by using the `DataLabelBase.Separator` property. The most frequently used separators are *comma*, *semicolon*, *period*, *new line* and *space*. However, you can specify a custom separator.

[Show Me](#)

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a pie chart and adjust the display settings of its data labels. In particular, set the `DataLabelBase.ShowCategoryName` and `DataLabelBase.ShowPercent` properties to **true** to display the category name and percentage value in a data label at the same time. To separate these items, assign a new line character to the `DataLabelBase.Separator` property, so the percentage value will be automatically wrapped to a new line.

C#

```
(DataLabelsActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.Pie, worksheet["B2:C7"]);
chart.TopLeftCell = worksheet.Cells["E2"];
chart.BottomRightCell = worksheet.Cells["K15"];
// Display the category name and percentage.
DataLabelOptions dataLabels = chart.Views[0].DataLabels;
dataLabels.ShowCategoryName = true;
dataLabels.ShowPercent = true;
dataLabels.Separator = "\n";
// Set the chart style.
chart.Style = ChartStyle.ColorGradient;
// Hide the legend.
chart.Legend.Visible = false;
// Set the angle of the first pie-chart slice.
chart.Views[0].FirstSliceAngle = 100;
```

Visual Basic

```
(DataLabelsActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.Pie, worksheet("B2:C7"))
chart.TopLeftCell = worksheet.Cells("E2")
chart.BottomRightCell = worksheet.Cells("K15")
' Display the category name and percentage.
Dim dataLabels As DataLabelOptions = chart.Views(0).DataLabels
dataLabels.ShowCategoryName = True
dataLabels.ShowPercent = True
dataLabels.Separator = ControlChars.Lf
' Set the chart style.
chart.Style = ChartStyle.ColorGradient
' Hide the legend.
chart.Legend.Visible = False
' Set the angle of the first pie-chart slice.
chart.Views(0).FirstSliceAngle = 100
```

Specify the Position of Data Labels

When there are many data points in a series or the label text is too long, the data labels may overlap, making the chart unreadable. To avoid this, you can adjust the positions of data labels by using the `DataLabelBase.LabelPosition` property. Depending on the chart type, assign one of the `DataLabelPosition` enumeration values to the **LabelPosition** property to specify the placement of data labels relative to their data markers.

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create the clustered column chart and display its data labels by setting the `DataLabelBase.ShowValue` property to **true**. To specify the location of data labels on the chart, use the `DataLabelBase.LabelPosition` property. In this example, the `DataLabelPosition.Center` value is used, so data labels will be displayed centered inside columns.

C#	
	<pre>(DataLabelsActions.cs) Worksheet worksheet = workbook.Worksheets["chartTask3"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Create a chart and specify its location. Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B chart.TopLeftCell = worksheet.Cells["H2"]; chart.BottomRightCell = worksheet.Cells["N14"]; // Display data labels and specify their position within the chart. chart.Views[0].DataLabels.ShowValue = true; chart.Views[0].DataLabels.LabelPosition = DataLabelPosition.Center;</pre>
Visual Basic	
	<pre>(DataLabelsActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask3") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works chart.TopLeftCell = worksheet.Cells("H2") chart.BottomRightCell = worksheet.Cells("N14") ' Display data labels and specify their position within the chart. chart.Views(0).DataLabels.ShowValue = True chart.Views(0).DataLabels.LabelPosition = DataLabelPosition.Center</pre>

Apply Number Format to Data Labels

You can specify how to display numeric values in data labels by applying number formats. A number in a data label can appear as a percentage, decimal, currency, scientific, fraction, text, accounting, date, time, or custom value. To apply a number format to data labels, utilize the `DataLabelBase.NumberFormat` property, which provides access to the `NumberFormatOptions` object containing format options for displaying numbers in different chart elements. Next, assign the corresponding number format code to the `NumberFormatOptions.FormatCode` property. Set the `NumberFormatOptions.IsSourceLinked` property to **false** to indicate that the applied format differs from the number format of the worksheet cell containing the data point value.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a clustered column chart and format numeric values displayed in data labels as percentage values. To apply the number

format, set the `NumberFormatOptions.IsSourceLinked` property to **false** and assign the corresponding format code to the `NumberFormatOptions.FormatCode` property.

C#

```
(DataLabelsActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B2:N14"]);
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Display data labels and specify their position within the chart.
chart.Views[0].DataLabels.ShowValue = true;
chart.Views[0].DataLabels.LabelPosition = DataLabelPosition.Center;
// Format data labels.
chart.Views[0].DataLabels.NumberFormat.FormatCode = "0%";
chart.Views[0].DataLabels.NumberFormat.IsSourceLinked = false;
```

Visual Basic

```
(DataLabelsActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B2:N14"])
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Display data labels and specify their position within the chart.
chart.Views(0).DataLabels.ShowValue = True
chart.Views(0).DataLabels.LabelPosition = DataLabelPosition.Center
' Format data labels.
chart.Views(0).DataLabels.NumberFormat.FormatCode = "0%"
chart.Views(0).DataLabels.NumberFormat.IsSourceLinked = False
```

Create a Custom Label Entry

Displaying labels for all data series in the current view can make the chart overcrowded. However, you can avoid this by displaying labels for a particular series or individual data points. An individual data label is defined by the `DataLabel` object that is stored in the collection of customized data labels. To display data labels for a specific series, access the `DataLabelCollection` by using the `Series.CustomDataLabels` property, then use the collection's properties to show and modify the data labels. To indicate that custom data labels are used, set the `Series.UseCustomDataLabels` property to **true**.

To display an individual data label, add a `DataLabel` instance to the `DataLabelCollection` collection with the index set to the index of the selected data point. Next, set the label's `DataLabelBase.ShowValue` property (or any other `DataLabelBase.Show*` property depending on the [information you wish to display in the label](#)) to **true**. You can also specify the custom text for a data label by using the `DataLabel.Text` property, which provides access to the `ChartText` object. Use the `ChartText.SetValue` method to specify a new text for a data label, or utilize the `ChartText.SetReference` method to retrieve the text for a label from the worksheet cell.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#

```
(DataLabelsActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Display the data label for the last point of the second series.
chart.Series[1].CustomDataLabels.Add(1).ShowValue = true;
chart.Series[1].UseCustomDataLabels = true;
```

Visual Basic

```
(DataLabelsActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Display the data label for the last point of the second series.
chart.Series(1).CustomDataLabels.Add(1).ShowValue = True
chart.Series(1).UseCustomDataLabels = True
```

See Also

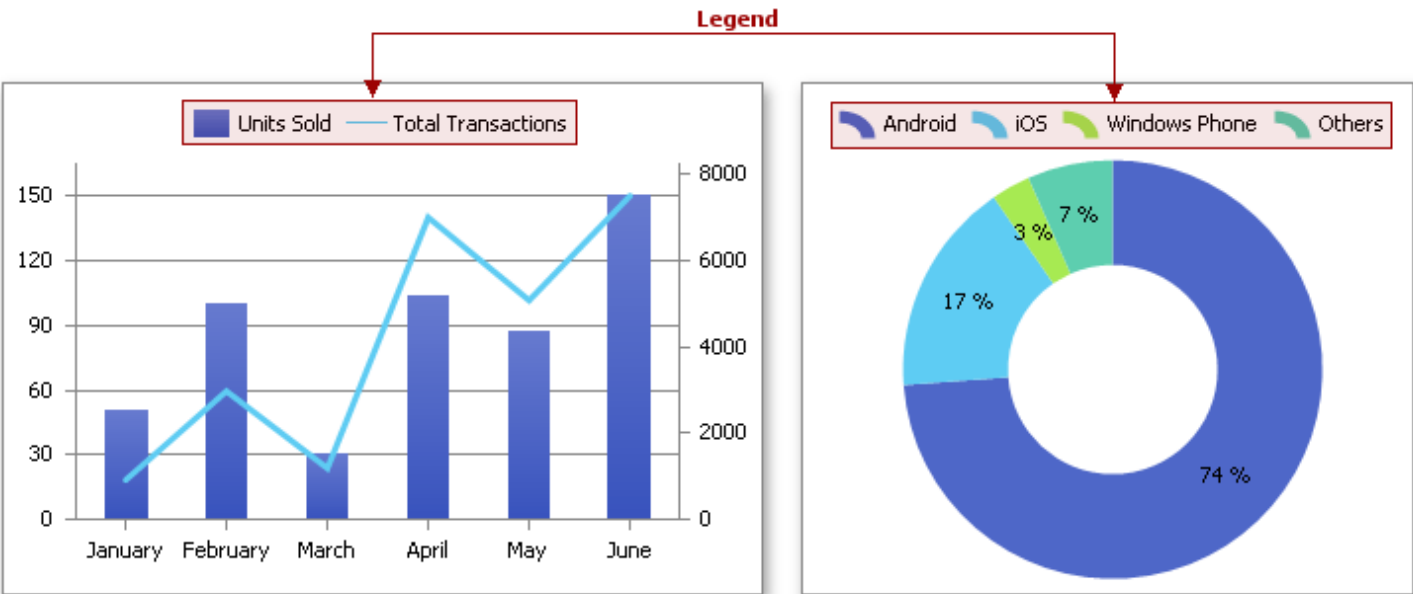
[How to: Create a Basic Chart](#)

[How to: Format Chart Elements](#)

How to: Show or Hide the Chart Legend

Office File API > Spreadsheet Document API > Examples > Charts > How to: Show or Hide the Chart Legend

After you [create a chart](#), its legend appears by default. A chart legend is a box that identifies data series displayed on a chart. In most cases, the legend displays series names, but in a *pie* or *doughnut* chart it shows data points of a single series. The legend also adds a sample of the line style, color and fill pattern used to draw the series on the chart.



The chart legend is defined by the Legend object, which can be accessed by utilizing the ChartObject.Legend property. To specify the legend placement, use the Legend.Position property. By default, the legend does not overlap the chart. However, to save space in the chart, you can turn this option off by setting the Legend.Overlay property to **true**. To remove the legend completely, set the Legend.Visible property to **false**.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#

```
(LegendActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B2:F6"]);
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Specify the position of the legend.
chart.Legend.Position = LegendPosition.Bottom;
```

Visual Basic

```
(LegendActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Specify the position of the legend.
chart.Legend.Position = LegendPosition.Bottom
```

Change a Legend Entry

You can also modify the individual legend entries by utilizing the `Legend.CustomEntries` property, which provides access to the collection of customized legend entries (`LegendEntryCollection`). For example, to hide a legend entry, add a `LegendEntry` instance to the collection with the index set to the index of the selected entry. Next, set the `LegendEntry.Hidden` property to **true**. You can also change the font attributes of an individual entry by utilizing the `LegendEntry.Font` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to remove the required entries from the chart legend by utilizing the `Legend.CustomEntries` property.

C#	
<pre>(LegendActions.cs) Worksheet worksheet = workbook.Worksheets["chartTask3"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Create a chart and specify its location. Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B chart.TopLeftCell = worksheet.Cells["H2"]; chart.BottomRightCell = worksheet.Cells["N14"]; // Exclude entries from the legend. chart.Legend.CustomEntries.Add(2).Hidden = true; chart.Legend.CustomEntries.Add(3).Hidden = true;</pre>	
Visual Basic	
<pre>(LegendActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask3") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works chart.TopLeftCell = worksheet.Cells("H2") chart.BottomRightCell = worksheet.Cells("N14") ' Exclude entries from the legend. chart.Legend.CustomEntries.Add(2).Hidden = True chart.Legend.CustomEntries.Add(3).Hidden = True</pre>	

See Also

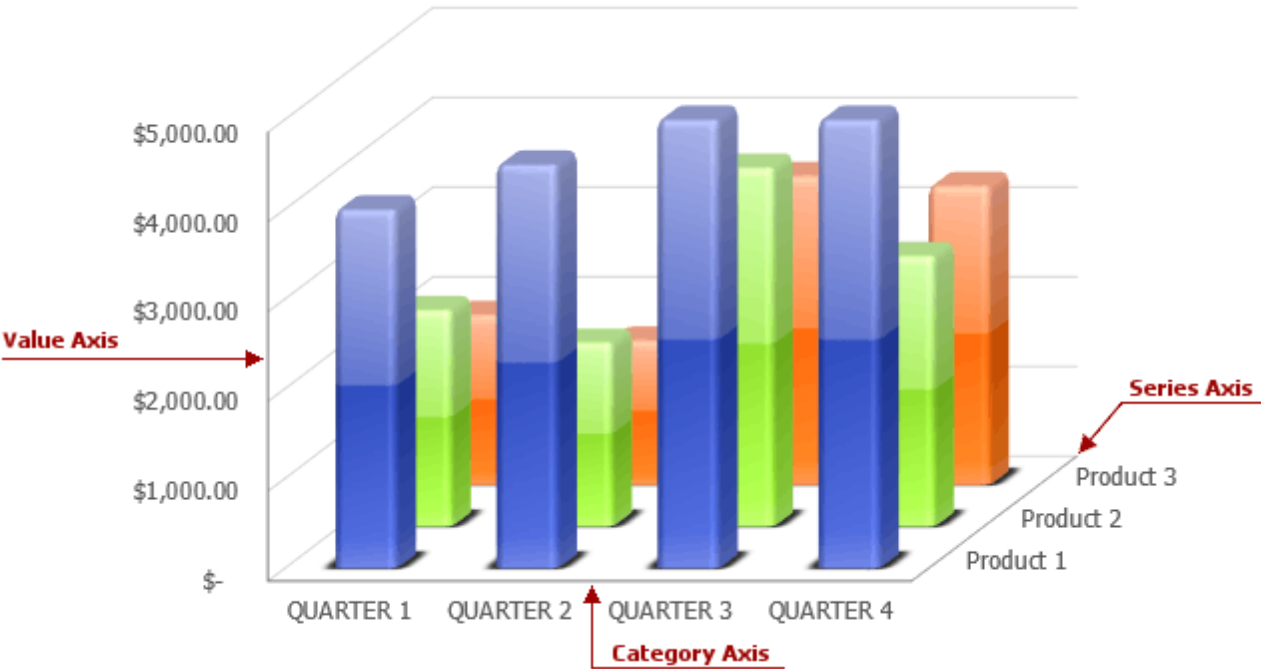
- [How to: Create a Basic Chart](#)
- [How to: Format Chart Elements](#)

How to: Change the Display of Chart Axes

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Charts](#) > [How to: Change the Display of Chart Axes](#)

When you [create a chart](#), its primary axes are generated automatically depending on the chart type. Most charts have two primary axes: the **category axis** (X-axis), usually running horizontally at the bottom of the plot area, and the **value axis** (Y-axis), usually running vertically on the left side of the plot area. 3-D charts also have the **depth** (or series) axis along which the series names are displayed. But some charts, like a *pie* or *doughnut* chart, have no axes at all.

If the chart type you selected supports axes, you can modify them by displaying axis tick marks and gridlines, changing the axis position, specifying the axis scaling, adding secondary axes, etc. You can also set the axis line settings and apply formatting options to the axis tick-mark labels as described in the [How to: Format Chart Elements](#) example.



Select the action you wish to perform.

- [Display or Hide Primary Axes](#)
- [Add Axis Titles](#)
- [Specify Scaling Options](#)
- [Format Numbers on the Axis](#)
- [Display Secondary Axes](#)
- [Display or Hide Axis Tick Marks and Gridlines](#)

Display or Hide Primary Axes

All primary axes are stored in the primary axis collection (AxisCollection), accessible using the ChartObject.PrimaryAxes property. An individual axis defined by the Axis object can be accessed using the indexer notation (**0** for the category axis, **1** for the value axis and **2** for the depth axis if it is present). To display or hide a primary axis, utilize the Axis.Visible property. To position an axis on the chart, use the Axis.Position property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a chart and specify its y-axis position using the Axis.Position property.

C#

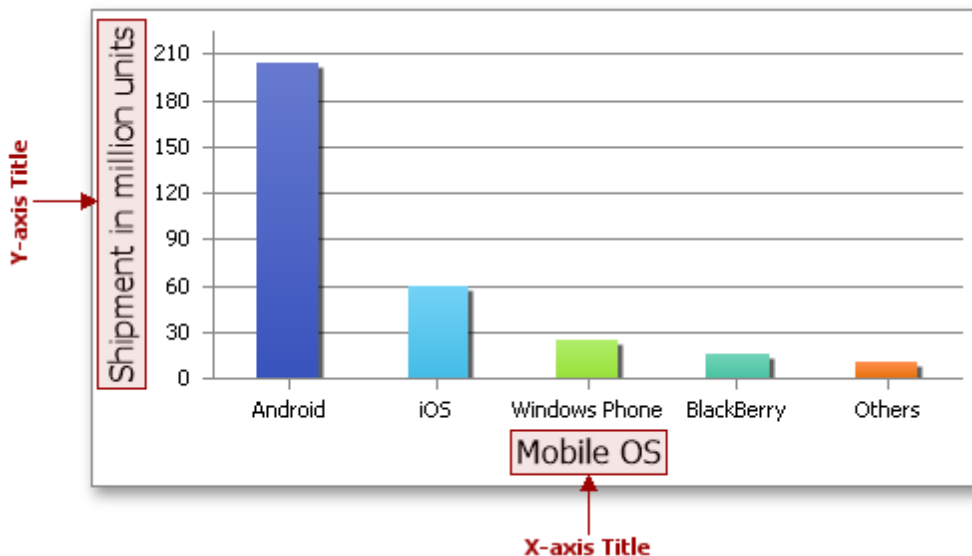
```
(AxesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B3:C5"]);
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Set the position of the value axis.
chart.PrimaryAxes[1].Position = AxisPosition.Right;
// Hide the legend.
chart.Legend.Visible = false;
```

Visual Basic

```
(AxesActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet("B3:C5"))
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Set the position of the value axis.
chart.PrimaryAxes(1).Position = AxisPosition.Right
' Hide the legend.
chart.Legend.Visible = False
```

Add Axis Titles

For both [primary](#) and [secondary](#) axes in the chart, it's possible to add and customize their text titles, which can be used to clarify data displayed along the axes.



The axis titles are not shown by default. However, you can add a title to any horizontal, vertical, or depth axes by utilizing the `Axis.Title` property, which accesses the `ChartTitleOptions` object containing display settings for axis titles. This object inherits the `ChartText` interface which provides methods used to specify the title text. Thus, to add the axis title, set the `ChartTitleOptions.Visible` property to **true**, and then do one of the following.

- To specify the title text manually, use the `ChartText.SetValue` method.
- To extract the text for the title from the cell range, use the `ChartText.SetReference` method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The following example demonstrates how to create a clustered bar chart and add the title to the value axis using the `Axis.Title` property, which returns the `ChartTitleOptions` object containing basic title options. Set the `ChartTitleOptions.Visible` property to **true** to display the axis title. To explicitly specify the text for the title, utilize the `ChartText.SetValue` method.

C#
<pre>(TitlesActions.cs) Worksheet worksheet = workbook.Worksheets["chartTask2"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Create a chart and specify its location. Chart chart = worksheet.Charts.Add(ChartType.BarClustered, worksheet["B4:C chart.TopLeftCell = worksheet.Cells["E3"]; chart.BottomRightCell = worksheet.Cells["K14"]; // Specify the axis title text. chart.PrimaryAxes[1].Title.Visible = true; chart.PrimaryAxes[1].Title.SetValue("Shipment in millions of units"); // Hide the legend. chart.Legend.Visible = false; // Specify that each data point in the series has a different color. chart.Views[0].VaryColors = true;</pre>

Visual Basic
<pre>(TitlesActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask2") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.BarClustered, workshee chart.TopLeftCell = worksheet.Cells("E3") chart.BottomRightCell = worksheet.Cells("K14") ' Specify the axis title text. chart.PrimaryAxes(1).Title.Visible = True chart.PrimaryAxes(1).Title.SetValue("Shipment in millions of units") ' Hide the legend. chart.Legend.Visible = False ' Specify that each data point in the series has a different color. chart.Views(0).VaryColors = True</pre>

Specify Scaling Options

By default, the minimum and maximum scale values for the numerical axis are evaluated automatically based on the data used by the chart. However, you can control the axis scaling by using the `Axis.Scaling` property, which gets the `AxisScaling` object. The object's properties allow you to control the minimum (`AxisScaling.Min`) and maximum (`AxisScaling.Max`) values of the axis, change the numerical axis to logarithmic (`AxisScaling.LogScale` and `AxisScaling.LogBase`) and specify the axis orientation (`AxisScaling.Orientation`).

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a clustered column chart and specify the value axis scale. Set the `AxisScaling.AutoMax` and `AxisScaling.AutoMin` properties to **false** to indicate that maximum and minimum values of the axis will be set manually by using the `AxisScaling.Max` and `AxisScaling.Min` properties

C#

```
(AxesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Set the minimum and maximum values for the chart value axis.
Axis axis = chart.PrimaryAxes[1];
axis.Scaling.AutoMax = false;
axis.Scaling.Max = 1;
axis.Scaling.AutoMin = false;
axis.Scaling.Min = 0;
// Hide the legend.
chart.Legend.Visible = false;
```

Visual Basic

```
(AxesActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Set the minimum and maximum values for the chart value axis.
Dim axis As Axis = chart.PrimaryAxes(1)
axis.Scaling.AutoMax = False
axis.Scaling.Max = 1
axis.Scaling.AutoMin = False
axis.Scaling.Min = 0
' Hide the legend.
chart.Legend.Visible = False
```

Format Numbers on the Axis

You can specify how to display numeric values on the axis by applying number formats. Numbers in the axis labels can appear as percentage, decimal, currency, scientific, fraction, text, accounting, date, time, or custom values. To apply a number format to tick-mark labels, utilize the `Axis.NumberFormat` property. This property provides access to the `NumberFormatOptions` object that contains format options for displaying numbers in chart elements. Assign the corresponding number format code to the

NumberFormatOptions.FormatCode property, and set the NumberFormatOptions.IsSourceLinked property to **false** to indicate that the applied format differs from the number format of the worksheet cell.

🔗 Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The following example demonstrates how to create a clustered column chart and display numbers on the value axis as percentage values. To apply the number format, set the NumberFormatOptions.IsSourceLinked property to **false** and assign the corresponding format code to the NumberFormatOptions.FormatCode property.

C#

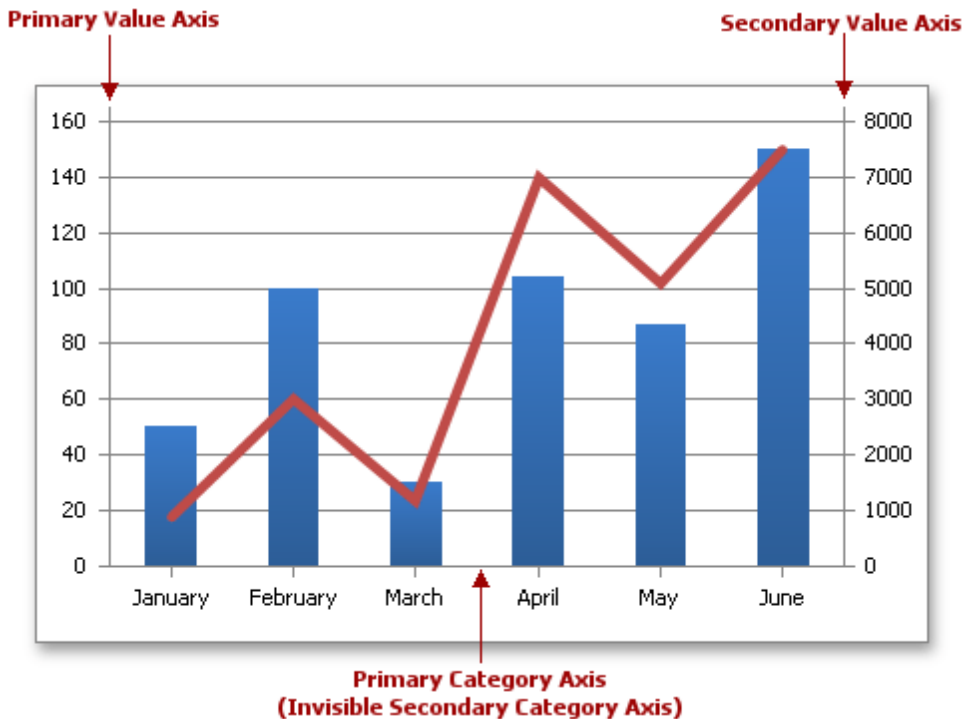
```
(AxesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Format the axis labels.
Axis axis = chart.PrimaryAxes[1];
axis.NumberFormat.FormatCode = "0%";
axis.NumberFormat.IsSourceLinked = false;
// Hide the legend.
chart.Legend.Visible = false;
```

Visual Basic

```
(AxesActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Format the axis labels.
Dim axis As Axis = chart.PrimaryAxes(1)
axis.NumberFormat.FormatCode = "0%"
axis.NumberFormat.IsSourceLinked = False
' Hide the legend.
chart.Legend.Visible = False
```

Display Secondary Axes

When data on a chart vary widely and the scale difference is huge, or when you combine different types of data on a combination chart, you can plot one or more data series along secondary axes.



To display secondary axes for a specific series, assign the `AxisGroup.Secondary` value to the `Series.AxisGroup` property. Note that you cannot make all the series on the chart secondary. An exception occurs when you try to change the axis group for the only primary series on a chart or for a series whose chart type does not allow secondary axes.

If the secondary axes were created successfully, they are added to the secondary axis collection (accessible using the `ChartObject.SecondaryAxes` property) and treated separately from the primary axes. You can get an individual secondary axis defined by the `Axis` object by its index in the collection (**0** for the secondary category axis and **1** for the secondary value axis). For the secondary axes, you can specify the same set of axis options that the primary axes have: for example, you can set the secondary axis position, adjust scaling options, add the axis title, etc.

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=T132724>.

The example below demonstrates how to display a secondary axis on a chart. The secondary axis group is used to plot data of a specific series, for which the `Series.AxisGroup` property is set to `AxisGroup.Secondary`.

C#

```
(SeriesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask5"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.LineMarker, worksheet["B2:D8"]
chart.TopLeftCell = worksheet.Cells["F2"];
chart.BottomRightCell = worksheet.Cells["L15"];
// Use the secondary axis.
chart.Series[1].AxisGroup = AxisGroup.Secondary;
// Specify the position of the legend.
chart.Legend.Position = LegendPosition.Top;
```

Visual Basic
<pre>(SeriesActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask5") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.LineMarker, worksheet chart.TopLeftCell = worksheet.Cells("F2") chart.BottomRightCell = worksheet.Cells("L15") ' Use the secondary axis. chart.Series(1).AxisGroup = AxisGroup.Secondary ' Specify the position of the legend. chart.Legend.Position = LegendPosition.Top</pre>

Display or Hide Axis Tick Marks and Gridlines

When you create a chart, it adds **major tick marks** to each axis by default. Tick marks are tiny lines used to indicate the demarcation of the axis. You can hide major tick marks or adjust their placement by using the Axis.MajorTickMarks property. To display and position **minor tick marks**, use the Axis.MinorTickMarks property. You can also specify how many tick marks should appear on the axis by changing the distance between the major and minor tick marks via the Axis.MajorUnit and Axis.MinorUnit properties.

Each major tick mark on the axis is accompanied by a tick-mark label - a textual identifier that explains the meaning of the axis units. Tick-mark labels are generated automatically based on the data displayed along the axis. However, you can remove axis labels from the chart by setting the Axis.TickLabelPosition property to AxisTickLabelPosition.None. You can also apply [number format to the numerical values](#) displayed in the labels.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create the clustered column chart and hide the axis major tick marks by setting the Axis.MajorTickMarks property to the AxisTickMarks.None value.

C#	
----	--

```
(AxesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Set the axis tick marks.
Axis axis = chart.PrimaryAxes[0];
axis.MajorTickMarks = AxisTickMarks.None;
axis = chart.PrimaryAxes[1];
axis.MajorTickMarks = AxisTickMarks.None;
// Hide the legend.
chart.Legend.Visible = false;
```

Visual Basic	
<pre>(AxesActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("chartTask3") workbook.Worksheets.ActiveWorksheet = worksheet ' Create a chart and specify its location. Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works chart.TopLeftCell = worksheet.Cells("H2") chart.BottomRightCell = worksheet.Cells("N14") ' Set the axis tick marks. Dim axis As Axis = chart.PrimaryAxes(0) axis.MajorTickMarks = AxisTickMarks.None axis = chart.PrimaryAxes(1) axis.MajorTickMarks = AxisTickMarks.None ' Hide the legend. chart.Legend.Visible = False</pre>	

To improve the readability of a chart, you can display **gridlines** - a series of horizontal and vertical lines running across the plot area. By default, the chart displays major gridlines for the primary value axis to help you to determine the value of each data point. However, you can hide major gridlines for the value axis or add gridlines to the category axis by using the `Axis.MajorGridlines` property. You can also display minor gridlines by utilizing the `Axis.MinorGridlines` property.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The example below demonstrates how to create a simple line chart and display major gridlines for the category axis and the minor gridlines for the value axis by setting the `ChartLineOptions.Visible` property to **true**.

C#	
----	--

```
(AxesActions.cs)
Worksheet worksheet = workbook.Worksheets["chartTask5"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.Line, worksheet["B2:C8"]);
chart.TopLeftCell = worksheet.Cells["F2"];
chart.BottomRightCell = worksheet.Cells["L15"];
// Display the major gridlines of the category axis.
chart.PrimaryAxes[0].MajorGridlines.Visible = true;
// Display the minor gridlines of the value axis.
chart.PrimaryAxes[1].MinorGridlines.Visible = true;
// Hide the legend.
chart.Legend.Visible = false;
```

Visual Basic

```
(AxesActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask5")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.Line, worksheet("B2:C8"))
chart.TopLeftCell = worksheet.Cells("F2")
chart.BottomRightCell = worksheet.Cells("L15")
' Display the major gridlines of the category axis.
chart.PrimaryAxes(0).MajorGridlines.Visible = True
' Display the minor gridlines of the value axis.
chart.PrimaryAxes(1).MinorGridlines.Visible = True
' Hide the legend.
chart.Legend.Visible = False
```

See Also

[How to: Create a Basic Chart](#)

[How to: Format Chart Elements](#)

How to: Protect a Chart

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Charts](#) > [How to: Protect a Chart](#)

After you [create a chart](#), you can apply chart protection to prevent your chart from being modified by a user. To protect a chart, utilize the ChartOptions.Protection property. After chart protection is specified, the chart becomes completely locked, so that a user cannot select the chart, modify its elements or change chart data references.

You can also protect the entire worksheet where the chart is located by using the Worksheet.Protect method. For detailed information on the protection functionality, refer to the [Protection](#) section.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

The following example demonstrates how to create a clustered column chart and apply chart protection using the ChartOptions.Protection property.

C#

```
(Protection.cs)
Worksheet worksheet = workbook.Worksheets["chartTask3"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a chart and specify its location.
Chart chart = worksheet.Charts.Add(ChartType.ColumnClustered, worksheet["B2:D4"]);
chart.TopLeftCell = worksheet.Cells["H2"];
chart.BottomRightCell = worksheet.Cells["N14"];
// Specify the chart style.
chart.Style = ChartStyle.ColorDark;
// Apply the chart protection.
chart.Options.Protection = ChartProtection.All;
```

Visual Basic

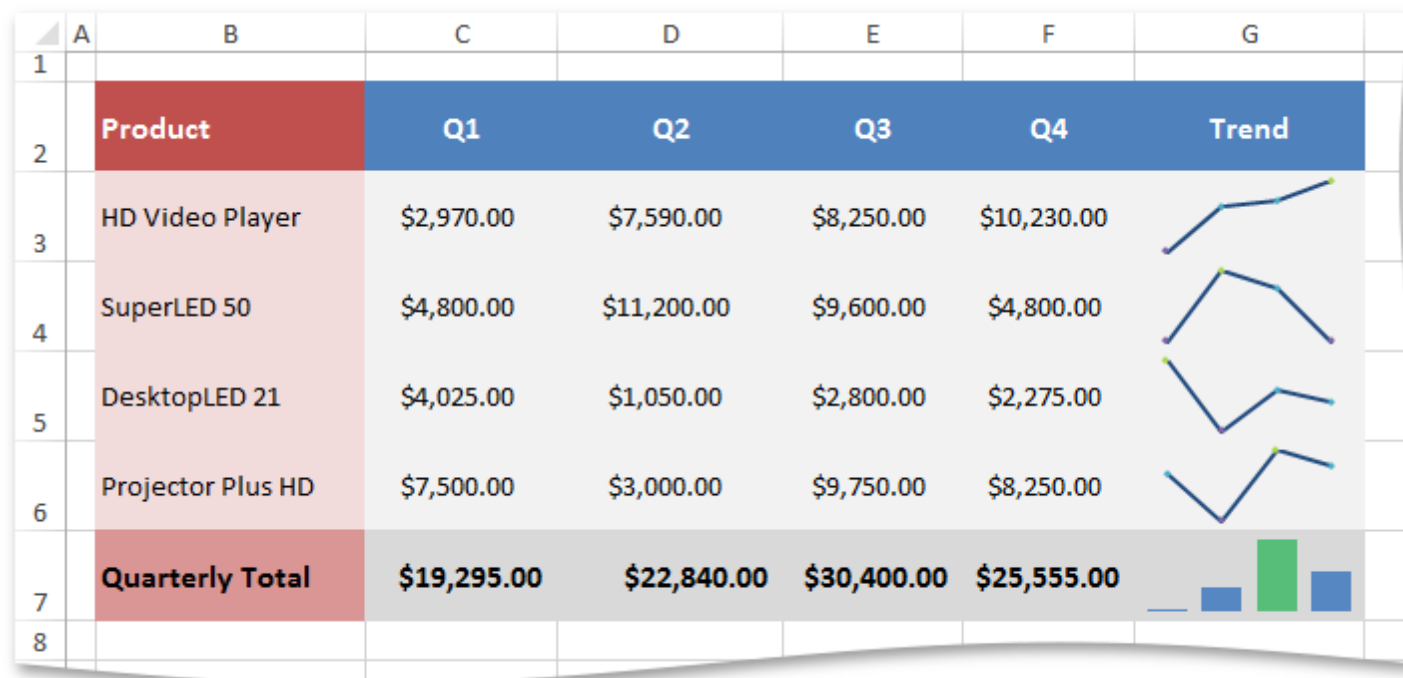
```
(Protection.vb)
Dim worksheet As Worksheet = workbook.Worksheets("chartTask3")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a chart and specify its location.
Dim chart As Chart = worksheet.Charts.Add(ChartType.ColumnClustered, works
chart.TopLeftCell = worksheet.Cells("H2")
chart.BottomRightCell = worksheet.Cells("N14")
' Specify the chart style.
chart.Style = ChartStyle.ColorDark
' Apply the chart protection.
chart.Options.Protection = ChartProtection.All
```

See Also
[How to: Create a Basic Chart](#)

Sparklines

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Sparklines](#)

This section contains examples that demonstrate how to insert *sparklines* in a workbook and adjust their settings. A sparkline is a tiny chart located in the background of a worksheet cell and used to visualize the variation of data over time. Sparklines are useful when you wish to perform an at-a-glance overview of data without needing to use the cumbersome charting functionality.






- [How to: Create Sparklines](#)
- [How to: Group and Ungroup Sparklines](#)
- [How to: Customize the Sparkline Appearance](#)
- [How to: Specify Sparkline Axis Settings](#)
- [How to: Delete Sparklines](#)

How to: Create Sparklines

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Sparklines](#) > [How to: Create Sparklines](#)

All sparklines in a worksheet are organized in groups. Each group can contain one or more sparklines of the same type that share identical formatting settings and axis scaling options. To create a new group of sparklines, get access to the worksheet's SparklineGroupCollection collection using the Worksheet.SparklineGroups property, and then call the SparklineGroupCollection.Add method. Pass the following parameters.

- A Range object that specifies a range of cells where the sparkline group should be located. You can either use a continuous one dimensional range, or specify each cell individually by inserting commas between two or more cell references within the Worksheet.Range property.
- A Range object that specifies the data source for the sparkline group. To add multiple sparklines at once, refer to multiple ranges within the Worksheet.Range property by inserting commas between two or more references, or use the IRangeProvider.Union method to combine multiple ranges into one complex range. Note that a data range for a single sparkline should be a continuous one-dimensional cell range (all cells of which are located in a single row or column). Also, the number of distinct ranges must be equal to the number of cells in the location range (defined by the SparklineGroup.Position property). If the specified cell range is insufficient for a sparkline group, an exception is thrown.
- A SparklineGroupType enumeration value that specifies the sparkline group type. The table below lists the available types of sparkline groups.

Sparkline Type	Enumeration Value	Description	Example
Line	SparklineGroupType.Line	Connects sparkline data points with a line.	
Column	SparklineGroupType.Column	Creates a series of columns, whose lengths are proportional to the data values they represent.	
Win/Loss	SparklineGroupType.Stacked	Creates a series of squares, each of which can occupy one of the following positions: - if the data value is positive , the square is displayed at the top of the cell (win); - if the value is negative , the square is displayed at the bottom of the cell (loss).	

To add a new sparkline to the existing group, access the collection of all sparklines contained in this group by using the SparklineGroup.Sparklines property, and call the SparklineCollection.Add method. This method allows you to define a cell at the intersection of the specified row and column where the new sparkline should be located and the cell range containing the source data for this sparkline.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#

```
(SparklineActions.cs)
Worksheet worksheet = workbook.Worksheets["SparklineExamples"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a group of line sparklines.
SparklineGroup quarterlyGroup = worksheet.SparklineGroups.Add(worksheet["G4:G6"], worksheet["C4:F4,C5:F5,C6:F6"]);
// Add one more sparkline to the existing group.
quarterlyGroup.Sparklines.Add(6, 6, worksheet["C7:F7"]);
// Display a column sparkline in the total cell.
SparklineGroup totalGroup = worksheet.SparklineGroups.Add(worksheet["G8"], worksheet["C8:F8"], SparklineGroupStyle.Column);
```

Visual Basic	
---------------------	--

```
(SparklineActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("SparklineExamples")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a group of line sparklines.
Dim quarterlyGroup As SparklineGroup = worksheet.SparklineGroups.Add(worksheet["G4:G6"], worksheet["C4:F4,C5:F5,C6:F6"], SparklineGroupStyle.Line)
' Add one more sparkline to the existing group.
quarterlyGroup.Sparklines.Add(6, 6, worksheet["C7:F7"])
' Display a column sparkline in the total cell.
Dim totalGroup As SparklineGroup = worksheet.SparklineGroups.Add(worksheet["G8"], worksheet["C8:F8"], SparklineGroupStyle.Column)
```

- See Also**
- [How to: Group and Ungroup Sparklines](#)
 - [How to: Customize the Sparkline Appearance](#)
 - [How to: Specify Sparkline Axis Settings](#)

How to: Group and Ungroup Sparklines

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Sparklines](#) > [How to: Group and Ungroup Sparklines](#)

The examples below demonstrate how to [group](#) and [ungroup](#) sparklines on a worksheet.

Group Sparklines

As mentioned in the [How to: Create Sparklines](#) article, all sparklines in a worksheet are stored in sparkline groups. A group allows you to apply formatting and scaling options to all sparklines in the group at the same time. However, you can regroup the existing sparklines to form new groups with their own custom settings. To rearrange worksheet sparklines, follow the steps below.

- 1. Get access to the sparklines you wish to regroup by their indices in the SparklineCollection collection accessible from the SparklineGroup.Sparklines property of the SparklineGroup objects containing the corresponding sparklines.
- 2. Call the SparklineGroupCollection.Add method overload and pass the following parameters: a list of the sparklines to be grouped together, and the SparklineGroupType enumeration member that specifies the type of the created group.

You can also move a single sparkline to another group by using the Sparkline.MoveTo method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

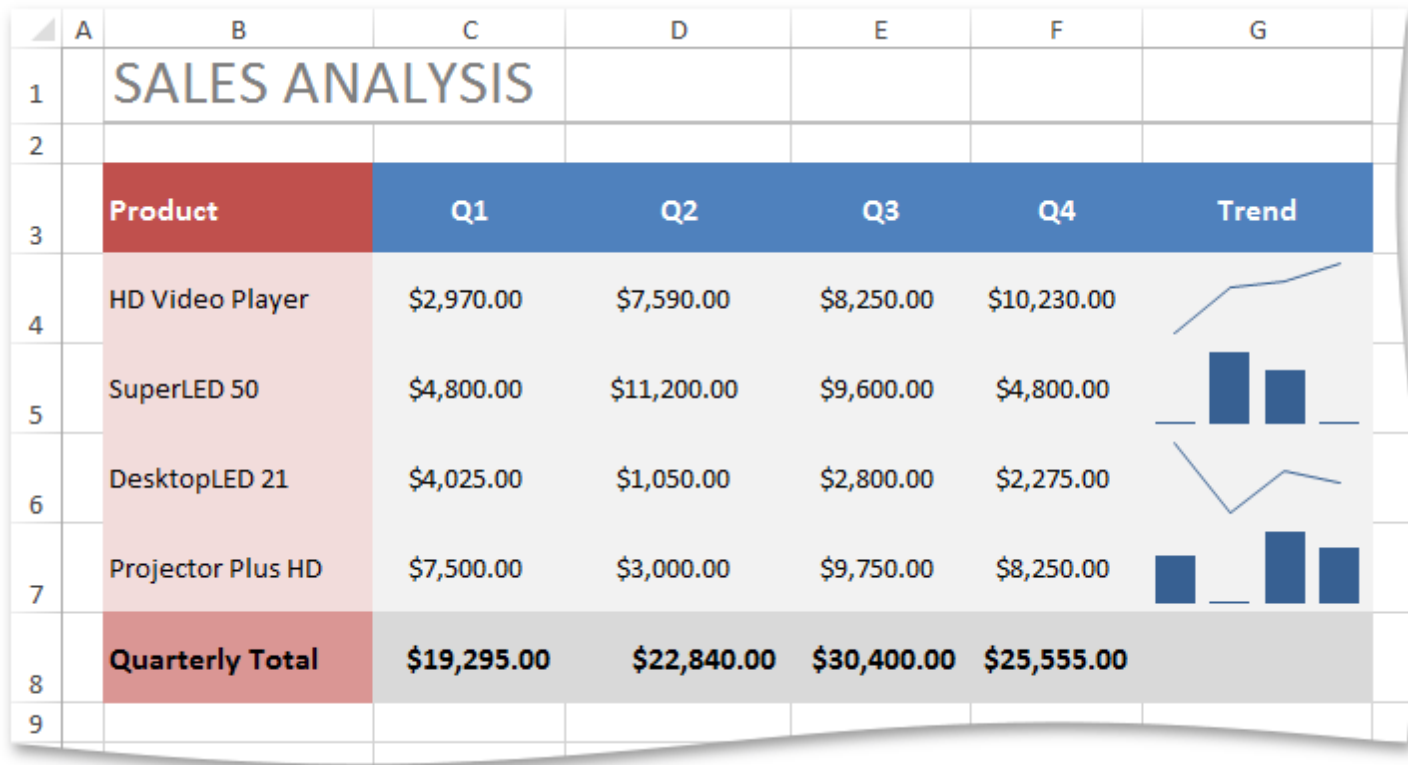
C#

```
(SparklineActions.cs)
Worksheet worksheet = workbook.Worksheets["SparklineExamples"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a group of line sparklines.
SparklineGroup lineGroup = worksheet.SparklineGroups.Add(worksheet["G4:G7"], worksheet["C4:F4,C5:F5,C6:F6"]);
// Rearrange sparklines by grouping the second and fourth sparklines together and changing the group type
Sparkline sparklineG5 = lineGroup.Sparklines[1];
Sparkline sparklineG7 = lineGroup.Sparklines[3];
SparklineGroup columnGroup = worksheet.SparklineGroups.Add(new List<Sparkline> { sparklineG5, sparklineG7},
```

Visual Basic

```
(SparklineActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("SparklineExamples")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a group of line sparklines.
Dim lineGroup As SparklineGroup = worksheet.SparklineGroups.Add(worksheet("G4:G7"), worksheet("C4:F4,C5:F5,C6:F6"))
' Rearrange sparklines by grouping the second and fourth sparklines together and changing the group type
Dim sparklineG5 As Sparkline = lineGroup.Sparklines(1)
Dim sparklineG7 As Sparkline = lineGroup.Sparklines(3)
Dim columnGroup As SparklineGroup = worksheet.SparklineGroups.Add(New List(Of Sparkline) (New Sparkline()
```

The image below shows the result (the workbook is opened in Microsoft® Excel®).



Ungroup Sparklines

To split a set of grouped sparklines into individual sparkline charts, use the SparklineGroup.UnGroup method. Calling this method will split the specified group into separate groups, each of which contains a single Sparkline object. All groups generated this way will have the same type, formatting and axis options as the source group.

C#
<pre>// Create a group of line sparklines. SparklineGroup lineGroup = worksheet.SparklineGroups.Add(worksheet["G4:G7"]) // Split the created group into individual sparklines. lineGroup.UnGroup();</pre>

Visual Basic
<pre>' Create a group of line sparklines. Dim lineGroup As SparklineGroup = worksheet.SparklineGroups.Add(worksheet(' Split the created group into individual sparklines. lineGroup.UnGroup()</pre>

See Also
[How to: Create Sparklines](#)

How to: Customize the Sparkline Appearance

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Sparklines](#) > [How to: Customize the Sparkline Appearance](#)

The example below demonstrates how to customize the sparkline appearance.

- Specify a sparkline color**
To specify a color used to draw each sparkline in the sparkline group, use the SparklineGroup.SeriesColor property. For a group of line sparklines, you can also set the line weight by utilizing the SparklineGroup.LineWeight property.

- Highlight data points**
You can highlight important data points on a sparkline by displaying and coloring markers on *line* sparklines, or by differentiating points by color on *column* and *win/loss* sparklines. To get access to the marker options, use the SparklineGroup.Points property, which returns the SparklinePoints object. The table below lists the object's properties used to show specific points on each sparkline in the sparkline group. All these properties return the SparklineColor object which controls the visibility and color of data markers.

Thus, to display the desired marker on a *line* sparkline, use the corresponding property to get access to the SparklineColor object, and set the SparklineColor.IsVisible property to **true**. In case of *column* and *win/loss* sparklines (which display all data markers by default), setting the SparklineColor.IsVisible property highlights the proper sparkline bars using the color defined by the applied sparkline style. To apply a custom color to the marker, assign the desired color to the SparklineColor.Color property.

Property	Description
SparklinePoints.Markers	Displays all markers on a line sparkline to differentiate individual data points.
SparklinePoints.Negative	Highlights markers that correspond to all data values less than zero.
SparklinePoints.Highest	Highlights a marker that corresponds to the highest data value.
SparklinePoints.Lowest	Highlights a marker that corresponds to the lowest data value.
SparklinePoints.First	Highlights a marker that corresponds to the first data point.
SparklinePoints.Last	Highlights a marker that corresponds to the last data point.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

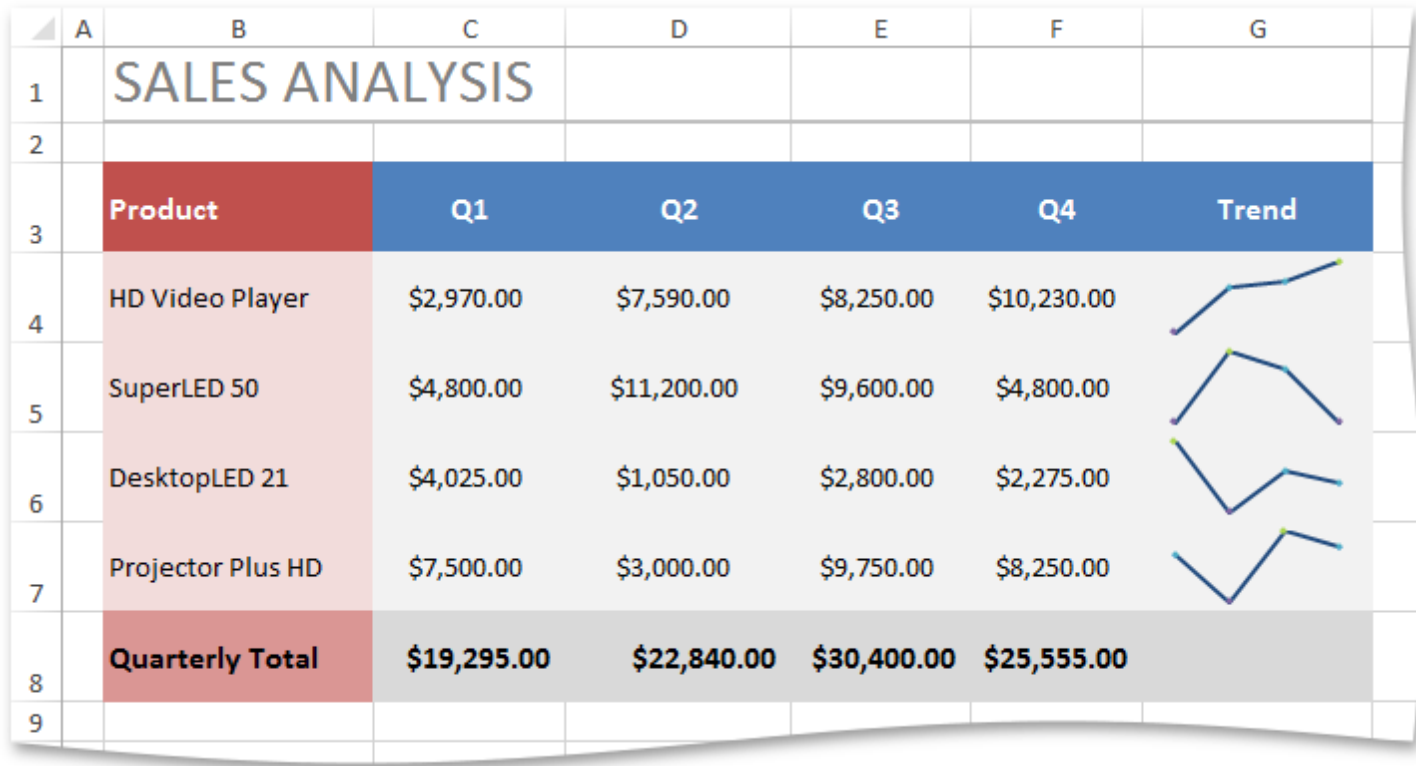
C#

```
(SparklineActions.cs)
Worksheet worksheet = workbook.Worksheets["SparklineExamples"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a group of line sparklines.
SparklineGroup lineGroup = worksheet.SparklineGroups.Add(worksheet["G4:G7"], worksheet["C4:F4,C5:F5,C6:F6"]);
// Customize the group appearance.
// Set the sparkline color.
lineGroup.SeriesColor = Color.FromArgb(0x1F, 0x49, 0x7D);
// Set the sparkline weight.
lineGroup.LineWeight = 1.5;
// Display data markers on the sparklines and specify their color.
SparklinePoints points = lineGroup.Points;
points.Markers.IsVisible = true;
points.Markers.Color = Color.FromArgb(0x4B, 0xAC, 0xC6);
// Highlight the highest and lowest points on each sparkline in the group.
points.Highest.Color = Color.FromArgb(0xA9, 0xD6, 0x4F);
points.Lowest.Color = Color.FromArgb(0x80, 0x64, 0xA2);
```

Visual Basic

```
(SparklineActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("SparklineExamples")
worksheet.Worksheets.ActiveWorksheet = worksheet
' Create a group of line sparklines.
Dim lineGroup As SparklineGroup = worksheet.SparklineGroups.Add(worksheet (
' Customize the group appearance.
' Set the sparkline color.
lineGroup.SeriesColor = Color.FromArgb(&H1F, &H49, &H7D)
' Set the sparkline weight.
lineGroup.LineWeight = 1.5
' Display data markers on the sparklines and specify their color.
Dim points As SparklinePoints = lineGroup.Points
points.Markers.IsVisible = True
points.Markers.Color = Color.FromArgb(&H4B, &HAC, &HC6)
' Highlight the highest and lowest points on each sparkline in the group.
points.Highest.Color = Color.FromArgb(&HA9, &HD6, &H4F)
points.Lowest.Color = Color.FromArgb(&H80, &H64, &HA2)
```

The following image shows sparkline charts created by the code above (the workbook is opened in Microsoft® Excel®).



See Also
[How to: Create Sparklines](#)

How to: Specify Sparkline Axis Settings

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Sparklines](#) > [How to: Specify Sparkline Axis Settings](#)

After you [add sparkline charts](#) to a worksheet, you can configure their [horizontal](#) and [vertical axis](#) settings to represent your data in the most efficient manner.

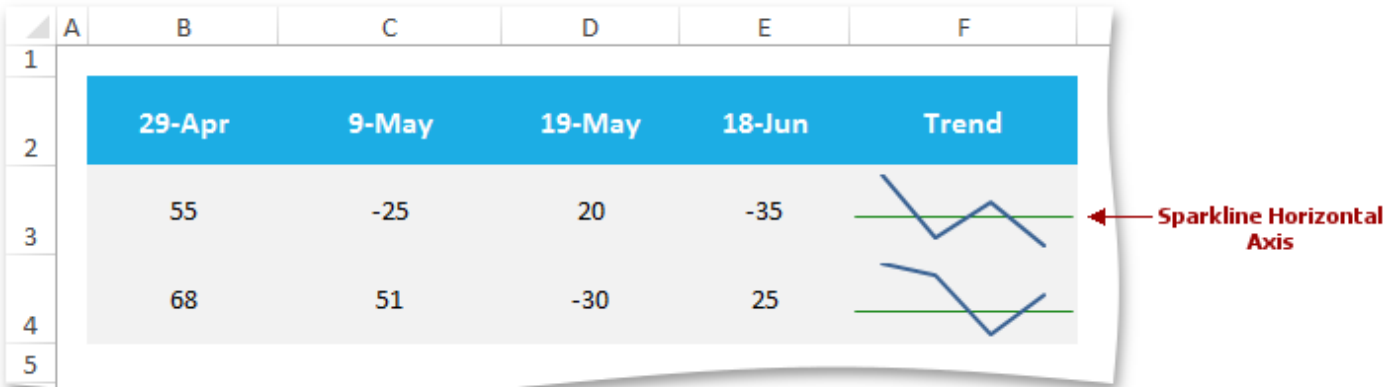
Horizontal Axis Options

To adjust the horizontal axis settings for a sparkline group, use the SparklineGroup.HorizontalAxis property. This property provides access to the SparklineHorizontalAxis object, which allows you to specify the following axis options.

- Visibility**
To display the horizontal axis on a sparkline, set the SparklineColor.IsVisible property of the SparklineHorizontalAxis object to **true**. Note that the horizontal axis for a sparkline is displayed only when the sparkline data contains both positive and negative values; otherwise, the axis is not shown.
To change the axis color, set the SparklineColor.Color property for the SparklineHorizontalAxis object.

```
C#  
  
// Create a group of line sparklines.  
SparklineGroup lineGroup = worksheet.SparklineGroups.Add(worksheet["F3:F4"], worksheet["B3:E3,B4:E4"], Sp  
// Display the horizontal axis and specify its color.  
lineGroup.HorizontalAxis.IsVisible = true;  
lineGroup.HorizontalAxis.Color = Color.ForestGreen;
```

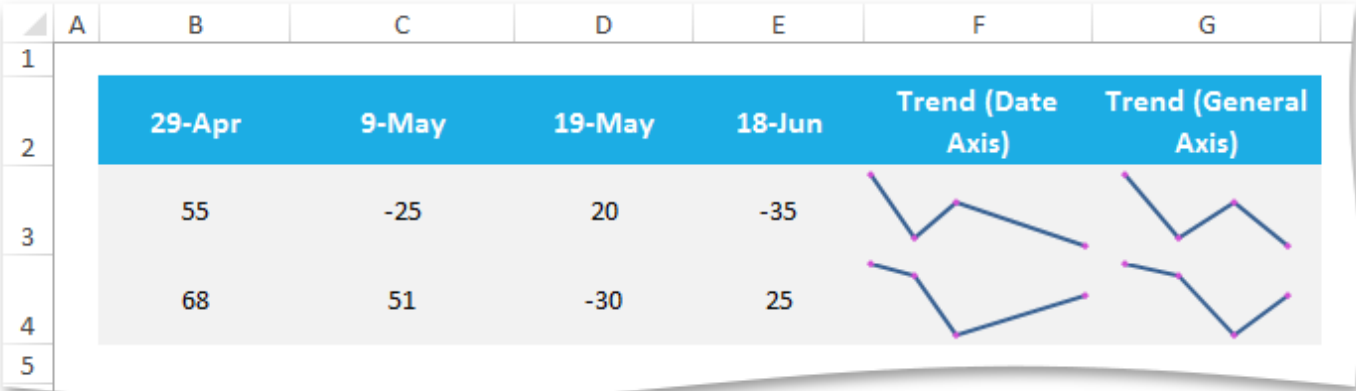
```
Visual Basic  
  
' Create a group of line sparklines.  
Dim lineGroup As SparklineGroup = worksheet.SparklineGroups.Add(worksheet("F3:F4"), worksheet("B3:E3,B4:E4  
' Display the horizontal axis and specify its color.  
lineGroup.HorizontalAxis.IsVisible = True  
lineGroup.HorizontalAxis.Color = Color.ForestGreen
```



- Axis Type**
By default, when you create a sparkline, it uses the general axis type that displays data points on a sparkline at regular intervals. However, if the underlying data occurs in irregular periods of time, you can reflect these time intervals on a sparkline by spacing data markers out in proportion to the date (the longer the interval between dates, the longer the space between the corresponding data markers on a sparkline). To change the axis type to date axis, assign the cell range containing date values for the required sparkline group to the SparklineGroup.DateRange property. This will automatically convert the horizontal axis to the date axis (the SparklineHorizontalAxis.IsDateAxis property will return **true**).

```
C#  
  
// Specify the date range for the sparkline group.  
lineGroup.DateRange = worksheet["B2:E2"];
```

```
Visual Basic  
  
' Specify the date range for the sparkline group.  
lineGroup.DateRange = worksheet("B2:E2")
```

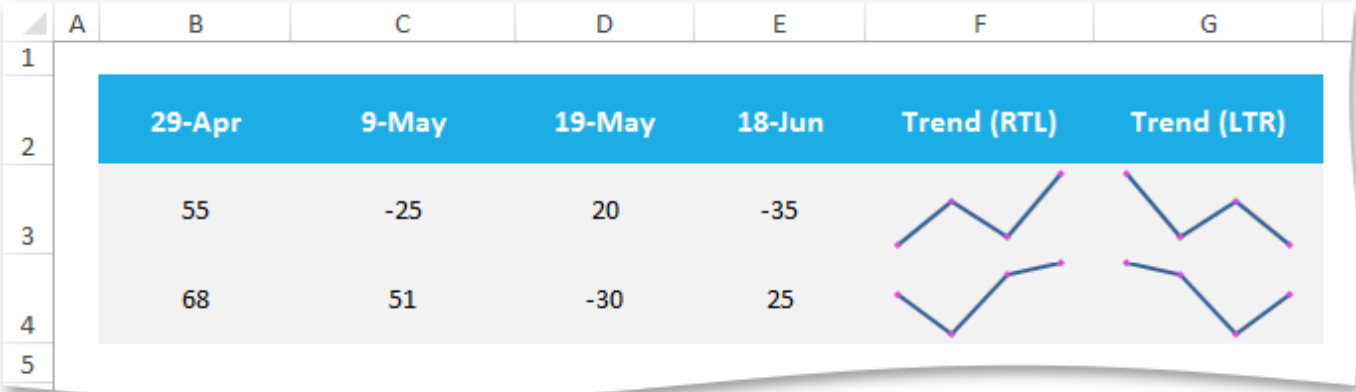


• Data Plotting Order

If you wish to display data points on each sparkline in the sparkline group in the reverse direction (in right-to-left order), set the SparklineHorizontalAxis.RightToLeft property to **true**.

```
C#
// Plot data in right-to-left order.
lineGroup.HorizontalAxis.RightToLeft = true;
```

```
Visual Basic
' Plot data in right-to-left order.
lineGroup.HorizontalAxis.RightToLeft = True
```



Vertical Axis Options

To specify scaling options for the vertical axis of a sparkline group, use the SparklineGroup.VerticalAxis property. This property allows you to specify how to calculate the minimum and maximum values for the vertical axis.

You can select one of the following options.

- **Automatic for each sparkline.** Minimum and maximum values are calculated individually for each sparkline in the group based on the lowest and highest values in the sparkline data range. To use this option, set the SparklineVerticalAxis.MinScaleType and SparklineVerticalAxis.MaxScaleType properties to the SparklineAxisScaling.Individual value.
- **Same for all sparklines.** All sparklines in the group use the same scale that is calculated automatically based on the lowest and highest values in the group data range. To use this option, set the SparklineVerticalAxis.MinScaleType and SparklineVerticalAxis.MaxScaleType properties to the SparklineAxisScaling.Group value.
- **Custom value.** Specifies custom values for the minimum and maximum of the vertical axis. To use custom scaling, set the SparklineVerticalAxis.MinScaleType and SparklineVerticalAxis.MaxScaleType properties to the SparklineAxisScaling.Custom value, and then assign the required numbers to the SparklineVerticalAxis.MinCustomValue and SparklineVerticalAxis.MaxCustomValue properties.

The example below demonstrates how to specify custom scaling values for the vertical axis of a sparkline group.

```
Show Me
```

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T132724>.

C#

```
(SparklineActions.cs)
Worksheet worksheet = workbook.Worksheets["SparklineExamples"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a group of column sparklines.
SparklineGroup columnGroup = worksheet.SparklineGroups.Add(worksheet["G4:G7"], worksheet["C4:F4,C5:F5,C6:F6"]);
// Specify the vertical axis options.
SparklineVerticalAxis verticalAxis = columnGroup.VerticalAxis;
// Set the custom minimum value for the vertical axis.
verticalAxis.MinScaleType = SparklineAxisScaling.Custom;
verticalAxis.MinCustomValue = 0;
// Set the custom maximum value for the vertical axis.
verticalAxis.MaxScaleType = SparklineAxisScaling.Custom;
verticalAxis.MaxCustomValue = 12000;
```

Visual Basic

```
(SparklineActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("SparklineExamples")
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a group of column sparklines.
Dim columnGroup As SparklineGroup = worksheet.SparklineGroups.Add(worksheet("G4:G7"), worksheet("C4:F4,C5:F5,C6:F6"))
' Specify the vertical axis options.
Dim verticalAxis As SparklineVerticalAxis = columnGroup.VerticalAxis
' Set the custom minimum value for the vertical axis.
verticalAxis.MinScaleType = SparklineAxisScaling.Custom
verticalAxis.MinCustomValue = 0
' Set the custom maximum value for the vertical axis.
verticalAxis.MaxScaleType = SparklineAxisScaling.Custom
verticalAxis.MaxCustomValue = 12000
```

See Also
[How to: Create Sparklines](#)

How to: Delete Sparklines

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Sparklines](#) > [How to: Delete Sparklines](#)

The examples below demonstrate how to remove [sparkline groups](#) or [individual sparklines](#) from a worksheet.

Delete a Sparkline Group

To remove a sparkline group, call the `SparklineGroupCollection.Remove` or `SparklineGroupCollection.RemoveAt` method of the `SparklineGroupCollection` object that is accessed from the `Worksheet.SparklineGroups` property.

C#

```
SparklineGroupCollection sparklineGroups = worksheet.SparklineGroups;
// Create a line sparkline group.
SparklineGroup lineGroup = sparklineGroups.Add(worksheet["G4:G7"], worksheet["C4:F4,C5:F5,C6:F6,C7:F7"], SparklineGroupType.Line);
// Create a column sparkline group.
SparklineGroup columnGroup = sparklineGroups.Add(worksheet["G8"], worksheet["C8:F8"], SparklineGroupType.Column);
// Delete the line sparkline group from the collection.
sparklineGroups.RemoveAt(0);
// Delete the column sparkline group from the collection.
sparklineGroups.Remove(columnGroup);
```

Visual Basic

```
Dim sparklineGroups As SparklineGroupCollection = worksheet.SparklineGroups
' Create a line sparkline group.
Dim lineGroup As SparklineGroup = sparklineGroups.Add(worksheet("G4:G7"), worksheet("C4:F4,C5:F5,C6:F6,C7:F7"), SparklineGroupType.Line)
' Create a column sparkline group.
Dim columnGroup As SparklineGroup = sparklineGroups.Add(worksheet("G8"), worksheet("C8:F8"), SparklineGroupType.Column)
' Delete the line sparkline group from the collection.
sparklineGroups.RemoveAt(0)
' Delete the column sparkline group from the collection.
sparklineGroups.Remove(columnGroup)
```

You can also use the `SparklineGroup.Delete` method to delete the required sparkline group from the collection.

C#

```
columnGroup.Delete();
```

Visual Basic

```
columnGroup.Delete()
```

To remove all sparkline groups from the worksheet at once, call the `SparklineGroupCollection.Clear` method.

C#

```
sparklineGroups.Clear();
```

Visual Basic

```
sparklineGroups.Clear()
```

Delete a Single Sparkline

To remove an individual sparkline (defined by the `Sparkline` object) from a sparkline group, call the `SparklineCollection.Remove` or `SparklineCollection.RemoveAt` method of the `SparklineCollection` collection accessed from the `SparklineGroup.Sparklines` property of the `SparklineGroup` object that stores the sparkline you wish to delete.

You can also use the `Sparkline.Delete` method to delete the required sparkline from the sparkline collection.

C#

```
// Create a group of line sparklines.
SparklineGroup lineGroup = worksheet.SparklineGroups.Add(worksheet["G4:G7"], worksheet["C4:F4,C5:F5,C6:F6,C7:F7"], SparklineGroupType.Line);
// Remove the first sparkline in the group.
lineGroup.Sparklines.RemoveAt(0);
// Remove the last sparkline in the group.
lineGroup.Sparklines[2].Delete();
```

Visual Basic

```
' Create a group of line sparklines.  
Dim lineGroup As SparklineGroup = worksheet.SparklineGroups.Add(worksheet("G4:G7"), worksheet("C4:F4,C5:F5"))  
' Remove the first sparkline in the group.  
lineGroup.Sparklines.RemoveAt(0)  
' Remove the last sparkline in the group.  
lineGroup.Sparklines(2).Delete()
```

See Also
[How to: Create Sparklines](#)

Formatting

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#)

This section contains the following examples:

- [How to: Apply a Style to a Cell or Range of Cells](#)
- [How to: Create or Modify a Style](#)
- [How to: Format a Cell or Range of Cells](#)
- [How to: Specify Number or Date Format for Cell Content](#)
- [How to: Change Cell Font and Background Color](#)
- [How to: Configure Cell Font Settings](#)
- [How to: Align Cell Content](#)
- [How to: Add and Remove Cell Borders](#)
- [How to: Clear Cell Formatting](#)

How to: Apply a Style to a Cell or Range of Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Apply a Style to a Cell or Range of Cells](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to format a cell, a range of cells, an entire row or an entire column by applying a [style](#).

1. Access the Style object that specifies the style to be applied to a cell or a range of cells. This style should be added to the [Workbook.Styles](#) collection.
2. Assign the required style object to the Range.Style property of the cell, cell range, row or column object.

C#

```
(FormattingActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
// Access the built-in "Good" MS Excel style from the Styles collection of the workbook.
Style styleGood = workbook.Styles[BuiltInStyleId.Good];
// Apply the "Good" style to a range of cells.
worksheet.Range["A1:C4"].Style = styleGood;
// Access a custom style that has been previously created in the loaded document by its name.
Style customStyle = workbook.Styles["Custom Style"];
// Apply the custom style to the cell.
worksheet.Cells["D6"].Style = customStyle;
// Apply the "Good" style to the eighth row.
worksheet.Rows[7].Style = styleGood;
// Apply the custom style to the "H" column.
worksheet.Columns["H"].Style = customStyle;
```

Visual Basic

```
(FormattingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Access the built-in "Good" MS Excel style from the Styles collection of
Dim styleGood As Style = workbook.Styles(BuiltInStyleId.Good)
' Apply the "Good" style to a range of cells.
worksheet.Range("A1:C4").Style = styleGood
' Access a custom style that has been previously created in the loaded doc
Dim customStyle As Style = workbook.Styles("Custom Style")
' Apply the custom style to the cell.
worksheet.Cells("D6").Style = customStyle
' Apply the "Good" style to the eighth row.
worksheet.Rows(7).Style = styleGood
' Apply the custom style to the "H" column.
worksheet.Columns("H").Style = customStyle
```

The image below shows how worksheet cells are formatted via styles (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F	G	H	I
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									

See Also

[How to: Clear Cell Formatting](#)

[How to: Create or Modify a Style](#)

How to: Create or Modify a Style

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Create or Modify a Style](#)

Create Your Own Custom Style

- Create a New Style

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

1. Add a new [style](#) to the [Workbook.Styles](#) collection by calling the `StyleCollection.Add` method with the style name passed as a parameter. This method returns the `Style` object corresponding to the created style. This object's `Style.Name` property is set to the specified name. Other settings are identical to the Normal built-in style.
2. Use properties of the newly created `Style` object to set the required format characteristics.

C#

```
(FormattingActions.cs)
// Add a new style under the "My Style" name to the Styles collection of the workbook.
Style myStyle = workbook.Styles.Add("My Style");
// Specify formatting characteristics for the style.
myStyle.BeginUpdate();
try {
    // Set the font color to Blue.
    myStyle.Font.Color = Color.Blue;
    // Set the font size to 12.
    myStyle.Font.Size = 12;
    // Set the horizontal alignment to Center.
    myStyle.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center;
    // Set the background.
    myStyle.Fill.BackgroundColor = Color.LightBlue;
    myStyle.Fill.PatternType = PatternType.LightGray;
    myStyle.Fill.PatternColor = Color.Yellow;
}
finally {
    myStyle.EndUpdate();
}
```

Visual Basic

```
(FormattingActions.vb)
' Add a new style under the "My Style" name to the Styles collection of the workbook.
Dim myStyle As Style = workbook.Styles.Add("My Style")
' Specify formatting characteristics for the style.
myStyle.BeginUpdate()
Try
    ' Set the font color to Blue.
    myStyle.Font.Color = Color.Blue
    ' Set the font size to 12.
    myStyle.Font.Size = 12
    ' Set the horizontal alignment to Center.
    myStyle.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center
    ' Set the background.
    myStyle.Fill.BackgroundColor = Color.LightBlue
    myStyle.Fill.PatternType = PatternType.LightGray
    myStyle.Fill.PatternColor = Color.Yellow
Finally
    myStyle.EndUpdate()
End Try
```

Create a New Style Based on a Built-In Style

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

1. Add a new [style](#) to the [Workbook.Styles](#) collection by calling the `StyleCollection.Add` method with the style name passed as a parameter. This method returns the `Style` object corresponding to the created style. This object's `Style.Name` property is set to the specified name. Other settings are identical to the Normal built-in style.
2. To copy all format settings from a Microsoft® Excel® built-in style to the newly created style, use the `Style.CopyFrom` method with the built-in style id (a member of the **BuiltInStyleId** enumeration) passed as a parameter.
3. Use properties of the newly created **Style** object to change the required format settings of the style.

C#
<pre>(FormattingActions.cs) // Add a new style under the "My Good Style" name to the Styles collection Style myGoodStyle = workbook.Styles.Add("My Good Style"); // Copy all format settings from the built-in Good style. myGoodStyle.CopyFrom(BuiltInStyleId.Good); // Modify the required formatting characteristics if needed. // ...</pre>

Visual Basic
<pre>(FormattingActions.vb) ' Add a new style under the "My Good Style" name to the Styles collection. Dim myGoodStyle As Style = workbook.Styles.Add("My Good Style") ' Copy all format settings from the built-in Good style. myGoodStyle.CopyFrom(BuiltInStyleId.Good) ' Modify the required formatting characteristics if needed. ' ...</pre>

Modify an Existing Style

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

1. Access a [style](#) to be modified. To do this, get the corresponding `Style` object from the [Workbook.Styles](#) collection by the style name or index.
2. Use the style's `Formatting.BeginUpdate` and `Formatting.EndUpdate` paired methods to modify the required format attributes of the style.

C#
<pre>(FormattingActions.cs) // Access the style to be modified. Style customStyle = workbook.Styles["Custom Style"]; // Change the required formatting characteristics of the style. customStyle.BeginUpdate(); try { customStyle.Fill.BackgroundColor = Color.Gold; // ... } finally { customStyle.EndUpdate(); }</pre>

Visual Basic	
<pre>(FormattingActions.vb) ' Access the style to be modified. Dim customStyle As Style = workbook.Styles("Custom Style") ' Change the required formatting characteristics of the style. customStyle.BeginUpdate() Try customStyle.Fill.BackgroundColor = Color.Gold ' ... Finally customStyle.EndUpdate() End Try</pre>	

You can also modify a style by accessing it directly from the cell or cell range to which this style is applied. To do this, use the Range.Style property. Style modifications will automatically be applied to all other cells that use this style.

C#	
<pre>// Modify the style applied to the "F10" cell. workbook.Worksheets[0].Cells["F10"].Style.Fill.BackgroundColor = Color.Sea // Modify the style applied to the "K8:M11" cell range. workbook.Worksheets[0].CreateRange("K8:M11").Style.Font.Color = Color.Red;</pre>	
Visual Basic	
<pre>' Modify the style applied to the "F10" cell. workbook.Worksheets(0).Cells("F10").Style.Fill.BackgroundColor = Color.Hot ' Modify the style applied to the "K8:M11" cell range. workbook.Worksheets(0).CreateRange("K8:M11").Style.Font.Color = Color.Red</pre>	

How to: Format a Cell or Range of Cells

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Format a Cell or Range of Cells](#)

Besides using [styles](#) that provide the capability to apply a predefined set of format attributes to multiple cells, you can also extend cell formatting with some explicit attributes. To perform direct cell formatting, use the Range object properties that are inherited from the Formatting interface (for example, Formatting.Fill, Formatting.Font, Formatting.Alignment and Formatting.Borders). Initially, these properties are set according to the style applied to the cell.

Thus, the actual cell appearance is determined by the format settings specified by the applied style and the cell's format settings. Each of these formatting types provides a set of *flags* (Formatting.Flags). Each flag corresponds to a specific group of format attributes. You can use these flags when formatting a cell, to control whether to use attributes specified in the applied style or attributes specified directly for the cell.

Group	Attributes	Flag
Alignment	Horizontal and vertical alignment of cell content, indentation, text wrap, text rotation and text shrinking.	StyleFlags.Alignment
Borders	Cell border line styles and colors.	StyleFlags.Borders
Fill	Cell background color and shading type.	StyleFlags.Fill
Font	Cell font settings (name, style, color and size).	StyleFlags.Font
Number Format	Cell number format.	StyleFlags.Number
Protection	Cell protection options (Locked and Hidden).	StyleFlags.Protection

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

- This example demonstrates how to format cells in a worksheet.
- To format an individual cell, access the corresponding Cell object and modify its properties (for example Formatting.Font, Formatting.Fill, Formatting.Borders, Formatting.Alignment and Formatting.NumberFormat).
 - To format a range of cells, access and modify the Formatting object via the Range.BeginUpdateFormatting - Range.EndUpdateFormatting paired methods.

C#

```
(FormattingActions.cs)
// Access the cell to be formatted.
Cell cell = worksheet.Cells["B2"];
// Specify font settings (font name, color, size and style).
cell.Font.Name = "MV Boli";
cell.Font.Color = Color.Blue;
cell.Font.Size = 14;
cell.Font.FontStyle = SpreadsheetFontStyle.Bold;
// Specify cell background color.
cell.Fill.BackgroundColor = Color.LightSkyBlue;
// Specify text alignment in the cell.
cell.Alignment.Vertical = SpreadsheetVerticalAlignment.Center;
cell.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center;
// Access the range of cells to be formatted.
Range range = worksheet.Range["C3:E6"];
// Begin updating of the range formatting.
Formatting rangeFormatting = range.BeginUpdateFormatting();
// Specify font settings (font name, color, size and style).
rangeFormatting.Font.Name = "MV Boli";
rangeFormatting.Font.Color = Color.Blue;
rangeFormatting.Font.Size = 14;
rangeFormatting.Font.FontStyle = SpreadsheetFontStyle.Bold;
// Specify cell background color.
rangeFormatting.Fill.BackgroundColor = Color.LightSkyBlue;
// Specify text alignment in cells.
rangeFormatting.Alignment.Vertical = SpreadsheetVerticalAlignment.Center;
rangeFormatting.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center;
// End updating of the range formatting.
range.EndUpdateFormatting(rangeFormatting);
```

Visual Basic	
--------------	--

```

(FormattinActions.vb)
' Access the cell to be formatted.
Dim cell As Cell = worksheet.Cells("B2")
' Specify font settings (font name, color, size and style).
cell.Font.Name = "MV Boli"
cell.Font.Color = Color.Blue
cell.Font.Size = 14
cell.Font.FontStyle = SpreadsheetFontStyle.Bold
' Specify cell background color.
cell.Fill.BackgroundColor = Color.LightSkyBlue
' Specify text alignment in the cell.
cell.Alignment.Vertical = SpreadsheetVerticalAlignment.Center
cell.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center
' Access the range of cells to be formatted.
Dim range As Range = worksheet.Range("C3:E6")
' Begin updating of the range formatting.
Dim rangeFormatting As Formatting = range.BeginUpdateFormatting()
' Specify font settings (font name, color, size and style).
rangeFormatting.Font.Name = "MV Boli"
rangeFormatting.Font.Color = Color.Blue
rangeFormatting.Font.Size = 14
rangeFormatting.Font.FontStyle = SpreadsheetFontStyle.Bold
' Specify cell background color.
rangeFormatting.Fill.BackgroundColor = Color.LightSkyBlue
' Specify text alignment in cells.
rangeFormatting.Alignment.Vertical = SpreadsheetVerticalAlignment.Center
rangeFormatting.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center
' End updating of the range formatting.
range.EndUpdateFormatting(rangeFormatting)

```

The image below shows formatted cells (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F
1						
2		Test				
3			Test	Test	Test	
4			Test	Test	Test	
5			Test	Test	Test	
6			Test	Test	Test	
7						
8						

See Also

[Formatting Cells](#)

[How to: Apply a Style to a Cell or Range of Cells](#)

[How to: Specify Number or Date Format for Cell Content](#)

[How to: Change Cell Font and Background Color](#)

[How to: Configure Cell Font Settings](#)
[How to: Align Cell Content](#)
[How to: Add and Remove Cell Borders](#)
[How to: Clear Cell Formatting](#)

How to: Specify Number or Date Format for Cell Content

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Specify Number or Date Format for Cell Content](#)

You can specify how to display [numeric values](#) in cells by applying *number formats*. For example, a number can appear in a cell as a percentage, decimal, currency, accounting, date or time value. To apply a number format to a cell or cell range, assign the corresponding number format code to the Formatting.NumberFormat property of the cell or cell range object. This document provides examples on how to apply different number formats to display cell content as [date and time](#) and as [percentage](#), [currency and decimal values](#), and how to [create custom number formats](#).

Date and Time Formats

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to specify different display formats for date and time values in cells.

C#

```
(FormattingActions.cs)
worksheet.Range["A1:F1"].Formula = "= Now() ";
// Apply different date display formats.
worksheet.Cells["A1"].NumberFormat = "m/d/yy";
worksheet.Cells["B1"].NumberFormat = "d-mmm-yy";
worksheet.Cells["C1"].NumberFormat = "dddd";
// Apply different time display formats.
worksheet.Cells["D1"].NumberFormat = "m/d/yy h:mm";
worksheet.Cells["E1"].NumberFormat = "h:mm AM/PM";
worksheet.Cells["F1"].NumberFormat = "h:mm:ss";
```

Visual Basic

```
(FormattingActions.vb)
worksheet.Range("A1:F1").Formula = "= Now() "
' Apply different date display formats.
worksheet.Cells("A1").NumberFormat = "m/d/yy"
worksheet.Cells("B1").NumberFormat = "d-mmm-yy"
worksheet.Cells("C1").NumberFormat = "dddd"
' Apply different time display formats.
worksheet.Cells("D1").NumberFormat = "m/d/yy h:mm"
worksheet.Cells("E1").NumberFormat = "h:mm AM/PM"
worksheet.Cells("F1").NumberFormat = "h:mm:ss"
```

The image below shows the different formats used to display date and time values in cells (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F
1	4/23/2013	23-Apr-13	Tuesday	4/23/2013 12:38	12:38 PM	12:38:13
2						

Number Formats

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>

This example demonstrates how to use different number formats to display numeric data using the most appropriate formats.

C#

```
(FormattingActions.cs)
// Display 111 as 111.
worksheet.Cells["A1"].Value = 111;
worksheet.Cells["A1"].NumberFormat = "#####";
// Display 222 as 00222.
worksheet.Cells["B1"].Value = 222;
worksheet.Cells["B1"].NumberFormat = "00000";
// Display 12345678 as 12,345,678.
worksheet.Cells["C1"].Value = 12345678;
worksheet.Cells["C1"].NumberFormat = "#,##";
// Display .126 as 0.13.
worksheet.Cells["D1"].Value = .126;
worksheet.Cells["D1"].NumberFormat = "0.##";
// Display 74.4 as 74.400.
worksheet.Cells["E1"].Value = 74.4;
worksheet.Cells["E1"].NumberFormat = "##.000";
// Display 1.6 as 160.0%.
worksheet.Cells["F1"].Value = 1.6;
worksheet.Cells["F1"].NumberFormat = "0.0%";
// Display 4321 as $4,321.00.
worksheet.Cells["G1"].Value = 4321;
worksheet.Cells["G1"].NumberFormat = "$#,##0.00";
// Display 8.75 as 8 3/4.
worksheet.Cells["H1"].Value = 8.75;
worksheet.Cells["H1"].NumberFormat = "# ?/?";
```

Visual Basic

```
(FormattingActions.vb)
' Display 111 as 111.
worksheet.Cells("A1").Value = 111
worksheet.Cells("A1").NumberFormat = "#####"
' Display 222 as 00222.
worksheet.Cells("B1").Value = 222
worksheet.Cells("B1").NumberFormat = "00000"
' Display 12345678 as 12,345,678.
worksheet.Cells("C1").Value = 12345678
worksheet.Cells("C1").NumberFormat = "#, #"
' Display .126 as 0.13.
worksheet.Cells("D1").Value = .126
worksheet.Cells("D1").NumberFormat = "0.##"
' Display 74.4 as 74.400.
worksheet.Cells("E1").Value = 74.4
worksheet.Cells("E1").NumberFormat = "##.000"
' Display 1.6 as 160.0%.
worksheet.Cells("F1").Value = 1.6
worksheet.Cells("F1").NumberFormat = "0.0%"
' Display 4321 as $4,321.00.
worksheet.Cells("G1").Value = 4321
worksheet.Cells("G1").NumberFormat = "$#,##0.00"
' Display 8.75 as 8 3/4.
worksheet.Cells("H1").Value = 8.75
worksheet.Cells("H1").NumberFormat = "# ?/?"
```

The image below shows the different formats used to display numbers in cells (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F	G	H
1	111	00222	12,345,678	0.13	74.400	160.0%	\$4,321.00	8 3/4
2								

Custom Number Formats

This example demonstrates how to create a custom number format to display positive numbers, negative numbers, zero and text values using different formats and colors. To do this, specify the appropriate code sections of the number format as required. These code sections are separated by semicolons and used in the following order.

<POSITIVE>;<NEGATIVE>;<ZERO>;<TEXT>

C#
// Set cell values. worksheet["A2:B2"].Value = -15.50; worksheet["A3:B3"].Value = 555; worksheet["A4:B4"].Value = 0; worksheet["A5:B5"].Value = "Name"; //Apply custom number format. worksheet["B2:B5"].NumberFormat = "[Green]#.00;[Red]#.00;[Blue]0.00;[Cyan]
Visual Basic

```
' Set cell values.
worksheet("A2:B2").Value = -15.50
worksheet("A3:B3").Value = 555
worksheet("A4:B4").Value = 0
worksheet("A5:B5").Value = "Name"
'Apply custom number format.
worksheet("B2:B5").NumberFormat = "[Green]#.00;[Red]#.00;[Blue]0.00;[Cyan]
```

The image below shows how different values are displayed in cells when a custom number format is applied (the workbook is opened in Microsoft® Excel®).

	A	B	
1	Value	Formatted Value	
2	-15.5	15.50	
3	555	555.00	
4	0	0.00	
5	Name	product: Name	
6			

See Also

- [Cell Data Types](#)
- [Dates and Times in Cells](#)

How to: Change Cell Font and Background Color

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Change Cell Font and Background Color](#)

You can change cell font color and cell background.

To specify these attributes for an [individual cell](#), modify the Cell object's Formatting.Font and Formatting.Fill properties, which are inherited from the Formatting interface.

To change color characteristics for a [range of cells](#), call the Range.BeginUpdateFormatting method for this range, modify the **Font** and **Fill** properties of the returned Formatting object and call the Range.EndUpdateFormatting method to finalize the modification.

• **Cell Font Color**

The Formatting.Font property returns the SpreadsheetFont object. Set this object's SpreadsheetFont.Color property to the required [Color](#) value to change cell font color.

• **Cell Background**

The Formatting.Fill property returns the Fill object. Use the following properties of this object to set cell background.

- Fill.BackgroundColor - sets the cell background color. It is also available via the Range.FillColor property of the cell or cell range object.
- Fill.PatternType - sets the type of cell background pattern. The available pattern types are accessed via the PatternType enumeration members.
- Fill.PatternColor - sets the shading color for a cell.

To share font color and background settings with multiple cells in a single step, [create or modify](#) a style with the Formatting.Font and Formatting.Fill properties specified as required, and assign this style to Range.Style for the desired cells.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to format color characteristics (font and background colors) for an individual cell and range of cells.

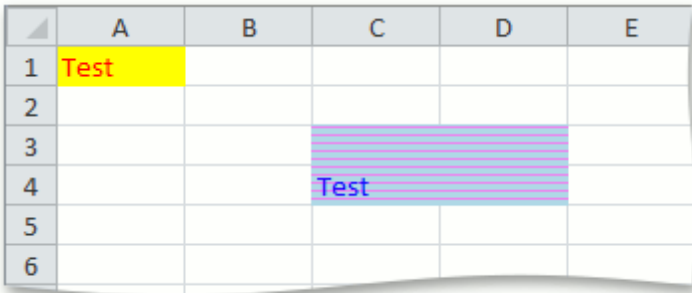
C#

```
(FormattingActions.cs)
// Format an individual cell.
worksheet.Cells["A1"].Font.Color = Color.Red;
worksheet.Cells["A1"].FillColor = Color.Yellow;
// Format a range of cells.
Range range = worksheet.Range["C3:D4"];
Formatting rangeFormatting = range.BeginUpdateFormatting();
rangeFormatting.Font.Color = Color.Blue;
rangeFormatting.Fill.BackgroundColor = Color.LightBlue;
rangeFormatting.Fill.PatternType = PatternType.LightHorizontal;
rangeFormatting.Fill.PatternColor = Color.Violet;
range.EndUpdateFormatting(rangeFormatting);
```

Visual Basic

```
(FormattingActions.vb)
' Format an individual cell.
worksheet.Cells("A1").Font.Color = Color.Red
worksheet.Cells("A1").FillColor = Color.Yellow
' Format a range of cells.
Dim range As Range = worksheet.Range("C3:D4")
Dim rangeFormatting As Formatting = range.BeginUpdateFormatting()
rangeFormatting.Font.Color = Color.Blue
rangeFormatting.Fill.BackgroundColor = Color.LightBlue
rangeFormatting.Fill.PatternType = PatternType.LightHorizontal
rangeFormatting.Fill.PatternColor = Color.Violet
range.EndUpdateFormatting(rangeFormatting)
```

The image below shows colored cells (the workbook is opened in Microsoft® Excel®).



	A	B	C	D	E
1	Test				
2					
3			Test		
4					
5					
6					

How to: Configure Cell Font Settings

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Configure Cell Font Settings](#)

To set different font attributes for cells, use the properties of the SpreadsheetFont object.

To access this object for modifying the font of an [individual cell](#), use the Cell object's Formatting.Font property that is inherited from the Formatting interface.

To change font characteristics for a [range of cells](#), call the Range.BeginUpdateFormatting method for this range, use the **Font** property of the returned Formatting object to access and modify the SpreadsheetFont object and call the Range.EndUpdateFormatting method to finalize the modification.

- The SpreadsheetFont object provides the following properties to change cell font attributes.
- SpreadsheetFont.Name - sets the cell font.
 - SpreadsheetFont.Size - sets the cell font size.
 - SpreadsheetFont.FontStyle, SpreadsheetFont.Bold, SpreadsheetFont.Italic - sets whether the cell font should be bold or italic.
 - SpreadsheetFont.Color - sets the cell font color.
 - SpreadsheetFont.Script - sets whether the cell text should be formatted as superscript or subscript.
 - SpreadsheetFont.Strikethrough - sets whether or not the cell text should be displayed with a horizontal line through the text.
 - SpreadsheetFont.UnderlineType - sets the type of underline applied to the font. The UnderlineType enumeration lists available underline types.

To share font settings with multiple cells in a single step, [create or modify](#) the style with the Formatting.Font property specified as required, and assign this style to Range.Style for the desired cells.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to format cell font characteristics (e.g., font name, size, color, style) by modifying the SpreadsheetFont object accessed via the Formatting.Font property of the Cell object.

C#

```
(FormattingActions.cs)
// Access the Font object.
SpreadsheetFont cellFont = worksheet.Cells["A1"].Font;
// Set the font name.
cellFont.Name = "Times New Roman";
// Set the font size.
cellFont.Size = 14;
// Set the font color.
cellFont.Color = Color.Blue;
// Format text as bold.
cellFont.Bold = true;
// Set font to be underlined.
cellFont.UnderlineType = UnderlineType.Double;
```

Visual Basic

```
(FormattingActions.vb)
' Access the Font object.
Dim cellFont As SpreadsheetFont = worksheet.Cells("A1").Font
' Set the font name.
cellFont.Name = "Times New Roman"
' Set the font size.
cellFont.Size = 14
' Set the font color.
cellFont.Color = Color.Blue
' Format text as bold.
cellFont.Bold = True
' Set font to be underlined.
cellFont.UnderlineType = UnderlineType.Double
```

The image below shows the specified cell font (the workbook is opened in Microsoft® Excel®).

	A	B	C
1	<u>Font Attributes</u>		
2			
3			
4			

How to: Align Cell Content

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Align Cell Content](#)

To align data contained within a cell, use the properties of the Alignment object.

To access this object to modify the alignment of [individual cell](#) content, use the Cell object's Formatting.Alignment property, which is inherited from the Formatting interface.

To change the alignment attributes for a [range of cells](#), call the Range.BeginUpdateFormatting method for this range, use the **Alignment** property of the returned Formatting object to access and modify the Alignment object, and call the Range.EndUpdateFormatting method to finalize the modification.

- The Alignment object provides the following properties to change cell alignment settings:
- Alignment.Horizontal, Alignment.Vertical - set the horizontal and vertical position of cell content.
 - Alignment.Indent - sets the indentation of cell content.
 - Alignment.WrapText - sets whether or not text should wrap in a cell.
 - Alignment.ShrinkToFit - sets whether or not text should shrink to fit cell size.
 - Alignment.RotationAngle - sets the rotation of text within a cell.

To share alignment settings with multiple cells in a single step, [create or modify](#) the style with the Formatting.Alignment property specified as required, and assign this style to Range.Style for the desired cells.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to specify the alignment of cell content by modifying the Alignment object accessed via the Formatting.Alignment property of the Cell object.

C#

```
(FormattingActions.cs)
Cell cellA1 = worksheet.Cells["A1"];
cellA1.Value = "Right and top";
cellA1.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Right;
cellA1.Alignment.Vertical = SpreadsheetVerticalAlignment.Top;
Cell cellA2 = worksheet.Cells["A2"];
cellA2.Value = "Center";
cellA2.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center;
cellA2.Alignment.Vertical = SpreadsheetVerticalAlignment.Center;
Cell cellA3 = worksheet.Cells["A3"];
cellA3.Value = "Left and bottom, indent";
cellA3.Alignment.Indent = 1;
Cell cellB1 = worksheet.Cells["B1"];
cellB1.Value = "The Alignment.ShrinkToFit property is applied";
cellB1.Alignment.ShrinkToFit = true;
Cell cellB2 = worksheet.Cells["B2"];
cellB2.Value = "Rotated Cell Contents";
cellB2.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center;
cellB2.Alignment.Vertical = SpreadsheetVerticalAlignment.Center;
cellB2.Alignment.RotationAngle = 15;
Cell cellB3 = worksheet.Cells["B3"];
cellB3.Value = "The Alignment.WrapText property is applied to wrap the text within a cell";
cellB3.Alignment.WrapText = true;
```

Visual Basic

```
(FormattingActions.vb)
Dim cellA1 As Cell = worksheet.Cells("A1")
cellA1.Value = "Right and top"
cellA1.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Right
cellA1.Alignment.Vertical = SpreadsheetVerticalAlignment.Top
Dim cellA2 As Cell = worksheet.Cells("A2")
cellA2.Value = "Center"
cellA2.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center
cellA2.Alignment.Vertical = SpreadsheetVerticalAlignment.Center
Dim cellA3 As Cell = worksheet.Cells("A3")
cellA3.Value = "Left and bottom, indent"
cellA3.Alignment.Indent = 1
Dim cellB1 As Cell = worksheet.Cells("B1")
cellB1.Value = "The Alignment.ShrinkToFit property is applied"
cellB1.Alignment.ShrinkToFit = True
Dim cellB2 As Cell = worksheet.Cells("B2")
cellB2.Value = "Rotated Cell Contents"
cellB2.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center
cellB2.Alignment.Vertical = SpreadsheetVerticalAlignment.Center
cellB2.Alignment.RotationAngle = 15
Dim cellB3 As Cell = worksheet.Cells("B3")
cellB3.Value = "The Alignment.WrapText property is applied to wrap the text within a cell"
cellB3.Alignment.WrapText = True
```

The image below shows how text can be aligned within worksheet cells (the workbook is opened in Microsoft® Excel®).

	A	B
1	Right and top	The Alignment.ShrinkToFit property is applied
2	Center	Rotated Cell Contents
3	Left and bottom, indent	The Alignment.WrapText property is applied to wrap the text within a cell

How to: Add and Remove Cell Borders

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Add and Remove Cell Borders](#)

To access and specify cell borders, use the Borders collection. To access this collection in an [individual cell](#), use the Cell object's Formatting.Borders property, which is inherited from the Formatting interface. To change the borders of a [range of cells](#), call the Range.BeginUpdateFormatting method for this range, and use the **Borders** property of the returned Formatting object to access and modify the Borders collection. Then call the Range.EndUpdateFormatting method to finalize the modification.

Each border of a cell is accessed via the corresponding property of the Borders collection and represented by the Border object. This object provides the Border.LineStyle and Border.Color properties, allowing you to specify border line style and color, respectively. The BorderLineStyle enumerator lists the available line styles. The border color is defined by the [Color](#) value.

Use the following properties and methods to specify each particular border of a cell or cell range, or set several borders at once (for example, all outside, inside, vertical or horizontal borders).

- Borders.BottomBorder - sets the bottom border of a cell or range of cells.
- Borders.LeftBorder - sets the left border of a cell or range of cells.
- Borders.RightBorder - sets the right border of a cell or range of cells.
- Borders.TopBorder - sets the top border of a cell or range of cells.
- Borders.DiagonalBorderType, Borders.DiagonalBorderLineStyle, Borders.DiagonalBorderColor or Borders.SetDiagonalBorders - set the diagonal borders of a single cell or each cell within a range of cells.
- Borders.InsideHorizontalBorders - sets all horizontal borders within a range of cells.
- Borders.InsideVerticalBorders - sets all vertical borders within a range of cells.
- Borders.SetOutsideBorders - sets all outside borders of a cell or range of cells.
- Range.SetInsideBorders - sets all inside borders within a range of cells.
- Borders.SetAllBorders - sets all outside borders for a single cell or all outside and inside borders for a range of cells.
- Borders.RemoveBorders - removes all borders of a cell or range of cells.

To share the same cell border settings between multiple cells in one step, [create or modify](#) a style with the Formatting.Borders property specified as required, and assign this style to Range.Style for the desired cells.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to specify different borders for individual cells and ranges of cells by modifying the Borders object.

C#

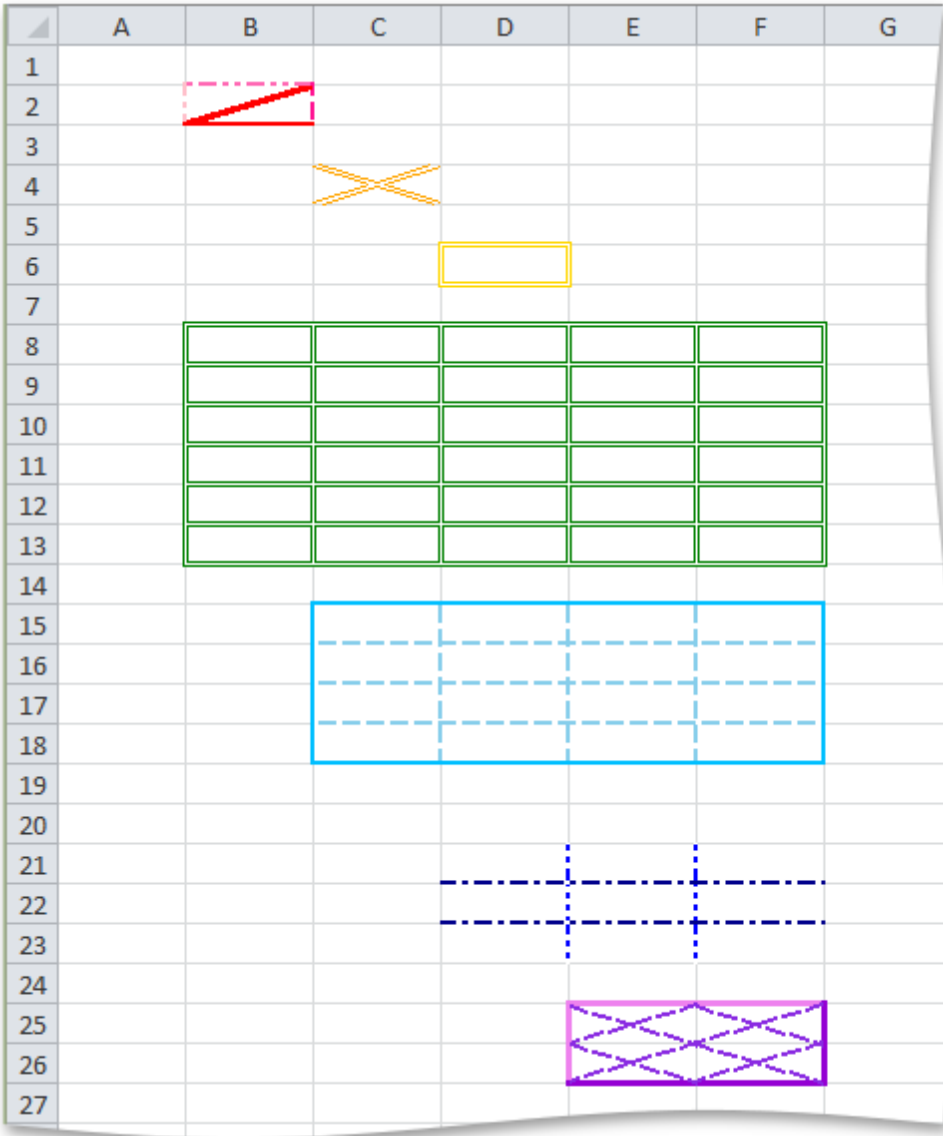
```
(FormattingActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
// Set each particular border for the cell.
Cell cellB2 = worksheet.Cells["B2"];
Borders cellB2Borders = cellB2.Borders;
cellB2Borders.LeftBorder.LineStyle = BorderLineStyle.MediumDashDot;
cellB2Borders.LeftBorder.Color = Color.Pink;
cellB2Borders.TopBorder.LineStyle = BorderLineStyle.MediumDashDotDot;
cellB2Borders.TopBorder.Color = Color.HotPink;
cellB2Borders.RightBorder.LineStyle = BorderLineStyle.MediumDashed;
cellB2Borders.RightBorder.Color = Color.DeepPink;
cellB2Borders.BottomBorder.LineStyle = BorderLineStyle.Medium;
cellB2Borders.BottomBorder.Color = Color.Red;
cellB2Borders.DiagonalBorderType = DiagonalBorderType.Up;
cellB2Borders.DiagonalBorderLineStyle = BorderLineStyle.Thick;
cellB2Borders.DiagonalBorderColor = Color.Red;
// Set diagonal borders for the cell.
Cell cellC4 = worksheet.Cells["C4"];
Borders cellC4Borders = cellC4.Borders;
cellC4Borders.SetDiagonalBorders(Color.Orange, BorderLineStyle.Double, DiagonalBorderType.UpAndDown);
// Set all outside borders for the cell in one step.
Cell cellD6 = worksheet.Cells["D6"];
cellD6.Borders.SetOutsideBorders(Color.Gold, BorderLineStyle.Double);
// Set all borders for the range of cells in one step.
Range range1 = worksheet.Range["B8:F13"];
range1.Borders.SetAllBorders(Color.Green, BorderLineStyle.Double);
// Set all inside and outside borders separately for the range of cells.
Range range2 = worksheet.Range["C15:F18"];
range2.SetInsideBorders(Color.SkyBlue, BorderLineStyle.MediumDashed);
range2.Borders.SetOutsideBorders(Color.DeepSkyBlue, BorderLineStyle.Medium);
// Set all horizontal and vertical borders separately for the range of cells.
Range range3 = worksheet.Range["D21:F23"];
Formatting range3Formatting = range3.BeginUpdateFormatting();
Borders range3Borders = range3Formatting.Borders;
range3Borders.InsideHorizontalBorders.LineStyle = BorderLineStyle.MediumDashDot;
range3Borders.InsideHorizontalBorders.Color = Color.DarkBlue;
range3Borders.InsideVerticalBorders.LineStyle = BorderLineStyle.MediumDashDotDot;
range3Borders.InsideVerticalBorders.Color = Color.Blue;
range3.EndUpdateFormatting(range3Formatting);
// Set each particular border for the range of cell.
Range range4 = worksheet.Range["E25:F26"];
Formatting range4Formatting = range4.BeginUpdateFormatting();
Borders range4Borders = range4Formatting.Borders;
range4Borders.SetOutsideBorders(Color.Black, BorderLineStyle.Thick);
range4Borders.LeftBorder.Color = Color.Violet;
range4Borders.TopBorder.Color = Color.Violet;
range4Borders.RightBorder.Color = Color.DarkViolet;
range4Borders.BottomBorder.Color = Color.DarkViolet;
range4Borders.DiagonalBorderType = DiagonalBorderType.UpAndDown;
range4Borders.DiagonalBorderLineStyle = BorderLineStyle.MediumDashed;
range4Borders.DiagonalBorderColor = Color.BlueViolet;
range4.EndUpdateFormatting(range4Formatting);
```

Visual Basic	
--------------	--

```
(FormattingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Set each particular border for the cell.
Dim cellB2 As Cell = worksheet.Cells("B2")
Dim cellB2Borders As Borders = cellB2.Borders
cellB2Borders.LeftBorder.LineStyle = BorderLineStyle.MediumDashDot
cellB2Borders.LeftBorder.Color = Color.Pink
cellB2Borders.TopBorder.LineStyle = BorderLineStyle.MediumDashDotDot
cellB2Borders.TopBorder.Color = Color.HotPink
cellB2Borders.RightBorder.LineStyle = BorderLineStyle.MediumDashed
cellB2Borders.RightBorder.Color = Color.DeepPink
cellB2Borders.BottomBorder.LineStyle = BorderLineStyle.Medium
cellB2Borders.BottomBorder.Color = Color.Red
cellB2Borders.DiagonalBorderType = DiagonalBorderType.Up
cellB2Borders.DiagonalBorderLineStyle = BorderLineStyle.Thick
cellB2Borders.DiagonalBorderColor = Color.Red
' Set diagonal borders for the cell.
Dim cellC4 As Cell = worksheet.Cells("C4")
Dim cellC4Borders As Borders = cellC4.Borders
cellC4Borders.SetDiagonalBorders(Color.Orange, BorderLineStyle.Double, Dia
' Set all outside borders for the cell in one step.
Dim cellD6 As Cell = worksheet.Cells("D6")
cellD6.Borders.SetOutsideBorders(Color.Gold, BorderLineStyle.Double)
' Set all borders for the range of cells in one step.
Dim range1 As Range = worksheet.Range("B8:F13")
range1.Borders.SetAllBorders(Color.Green, BorderLineStyle.Double)
' Set all inside and outside borders separately for the range of cells.
Dim range2 As Range = worksheet.Range("C15:F18")
range2.SetInsideBorders(Color.SkyBlue, BorderLineStyle.MediumDashed)
range2.Borders.SetOutsideBorders(Color.DeepSkyBlue, BorderLineStyle.Medium
' Set all horizontal and vertical borders separately for the range of cell
Dim range3 As Range = worksheet.Range("D21:F23")
Dim range3Formatting As Formatting = range3.BeginUpdateFormatting()
Dim range3Borders As Borders = range3Formatting.Borders
range3Borders.InsideHorizontalBorders.LineStyle = BorderLineStyle.MediumDa
range3Borders.InsideHorizontalBorders.Color = Color.DarkBlue
range3Borders.InsideVerticalBorders.LineStyle = BorderLineStyle.MediumDash
range3Borders.InsideVerticalBorders.Color = Color.Blue
range3.EndUpdateFormatting(range3Formatting)
' Set each particular border for the range of cell.
Dim range4 As Range = worksheet.Range("E25:F26")
Dim range4Formatting As Formatting = range4.BeginUpdateFormatting()
Dim range4Borders As Borders = range4Formatting.Borders
range4Borders.SetOutsideBorders(Color.Black, BorderLineStyle.Thick)
range4Borders.LeftBorder.Color = Color.Violet
range4Borders.TopBorder.Color = Color.Violet
range4Borders.RightBorder.Color = Color.DarkViolet
range4Borders.BottomBorder.Color = Color.DarkViolet
range4Borders.DiagonalBorderType = DiagonalBorderType.UpAndDown
range4Borders.DiagonalBorderLineStyle = BorderLineStyle.MediumDashed
range4Borders.DiagonalBorderColor = Color.BlueViolet
```

```
range4.EndUpdateFormatting(range4Formatting)
```

The image below shows which borders are applied to cells after executing the code above (the workbook is opened in Microsoft® Excel®).



How to: Clear Cell Formatting

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Formatting Cells](#) > [How to: Clear Cell Formatting](#)

This example demonstrates how to remove all [formatting](#) from a cell or range of cells. You can do this in one of the following ways.

- Call the `Worksheet.ClearFormats` method with the passed object specifying the cell or cell range to be cleared.
- [Apply](#) the *Normal* style to a cell or range of cells via the `Range.Style` property.

The *Normal* style object can be accessed from the [Workbook.Styles](#) collection by the style name (*Normal*) or index (by default, this style is added as the first in the collection of styles and cannot be deleted), or via the `StyleCollection.DefaultStyle` property.

C#

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
Worksheet worksheet = workbook.Worksheets[0];
// Call the ClearFormats method.
worksheet.ClearFormats(worksheet.Range["A1:C4"]);
// Apply the Normal style to cells.
worksheet.Cells["D6"].Style = workbook.Styles[0];
worksheet.Range["F3:H4"].Style = workbook.Styles[BuiltInStyleId.Normal];
worksheet.Rows[7].Style = workbook.Styles["Normal"];
worksheet.Columns["K"].Style = workbook.Styles.DefaultStyle;
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Call the ClearFormats method.
worksheet.ClearFormats(worksheet.Range("A1:C4"))
' Apply the Normal style to cells.
worksheet.Cells("D6").Style = workbook.Styles(0)
worksheet.Range("F3:H4").Style = workbook.Styles(BuiltInStyleId.Normal)
worksheet.Rows(7).Style = workbook.Styles("Normal")
worksheet.Columns("K").Style = workbook.Styles.DefaultStyle
```

Conditional Formatting

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#)

This section contains examples that demonstrate how to use conditional formatting to highlight important information and make data interpretation easier. Conditional formats such as cell background color, border line style or font color can be applied to cells whose values meet a certain condition specified by a conditional formatting rule.

	A	C	D	E	F	G
2		Country	Exports	Imports	Balance	Balance Ch
3		Australia	2,277.90	815.40	▲ 1,462.50	-0.12%
4		Belgium	2,808.90	1,826.20	▲ 982.70	0.13%
5		Brazil	3,423.20	2,571.40	▲ 851.80	-0.40%
6		Canada	26,453.10	28,343.10	▼ -1,890.00	-0.46%
7		China	8,786.50	36,646.20	▼ -27,859.70	0.14%
8		France	2,642.30	3,563.70	▼ -921.40	0.00%
9		Germany	4,052.50	9,888.30	▼ -5,835.80	0.04%
10		Hong Kong	3,385.40	413.00	▲ 2,972.40	-0.26%
11		India	1,935.80	4,201.50	▼ -2,265.70	3.20%
12		Italy	1,581.10	3,292.10	▼ -1,711.00	0.04%
13		Japan	5,785.20	11,190.40	▼ -5,405.20	-0.07%
14		Switzerland	2,150.80	3,249.10	▼ -1,098.30	2.93%

- [How to: Format Cell Values that are Above or Below the Average](#)
- [How to: Format Cells that are Between or Not Between Two Values](#)
- [How to: Format Top or Bottom Ranked Values](#)
- [How to: Format Cells based on the Text in the Cell](#)
- [How to: Format Unique or Duplicate Values, Blank Cells and Formula Errors](#)
- [How to: Format Cells that Contain Dates](#)
- [How to: Format Cells that are Less than, Greater than or Equal to a Value](#)
- [How to: Use a Formula to Determine which Cells to Format](#)
- [How to: Format Cells Using a Two-Color Scale](#)
- [How to: Format Cells Using a Three-Color Scale](#)
- [How to: Format Cells Using Data Bars](#)
- [How to: Format Cells Using Icon Sets](#)

How to: Format Cell Values that are Above or Below the Average

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cell Values that are Above or Below the Average](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to apply an "above or below average" conditional formatting rule to a range of cells.

1. To create a new conditional formatting rule represented by the `AverageConditionalFormatting` object, access the collection of conditional formats from the `Worksheet.ConditionalFormattings` property and call the `ConditionalFormattingCollection.AddAverageConditionalFormatting` method. Pass the following parameters:
 - A `Range` object that defines a range of cells to which the rule is applied.
 - A condition specified by one of the `ConditionalFormattingAverageCondition` enumeration values.
- If you use the overloaded method with three parameters, pass the third parameter:
- An integer value representing a number of standard deviations to be included above or below the mean in a range of cells.
2. Specify formatting options to be applied to cells if the condition is true using the `ISupportsFormatting.Formatting` property of the `AverageConditionalFormatting` object. Set the background color and specify the font attributes.

To remove the `AverageConditionalFormatting` object, use the `ConditionalFormattingCollection.Remove`, `ConditionalFormattingCollection.RemoveAt` or `ConditionalFormattingCollection.Clear` methods.

C#

```
(ConditionalFormatting.cs)
ConditionalFormattingCollection conditionalFormattings = worksheet.ConditionalFormattings;
// Create the rule highlighting values that are above the average in cells C2 through C15.
AverageConditionalFormatting cfRule1 = conditionalFormattings.AddAverageConditionalFormatting(worksheet["C2:C15"]);
// Specify formatting options to be applied to cells if the condition is true.
// Set the background color to yellow.
cfRule1.Formatting.Fill.BackgroundColor = Color.FromArgb(255, 0xFA, 0xF7, 0xAA);
// Set the font color to red.
cfRule1.Formatting.Font.Color = Color.Red;
// Create the rule highlighting values that are one standard deviation below the mean in cells D2 through D15.
AverageConditionalFormatting cfRule2 = conditionalFormattings.AddAverageConditionalFormatting(worksheet["D2:D15"]);
// Specify formatting options to be applied to cells if the conditions is true.
// Set the background color to light-green.
cfRule2.Formatting.Fill.BackgroundColor = Color.FromArgb(255, 0x9F, 0xFB, 0x69);
// Set the font color to blue-violet.
cfRule2.Formatting.Font.Color = Color.BlueViolet;
```

Visual Basic

```

(ConditionalFormatting.vb)
Dim conditionalFormattings As ConditionalFormattingCollection = worksheet.
' Create the rule highlighting values that are above the average in cells
Dim cfRule1 As AverageConditionalFormatting = conditionalFormattings.AddAv
' Specify formatting options to be applied to cells if the condition is tr
' Set the background color to yellow.
cfRule1.Formatting.Fill.BackgroundColor = Color.FromArgb(255, &HFA, &HF7,
' Set the font color to red.
cfRule1.Formatting.Font.Color = Color.Red
' Create the rule highlighting values that are one standard deviation belo
Dim cfRule2 As AverageConditionalFormatting = conditionalFormattings.AddAv
' Specify formatting options to be applied to cells if the conditions is t
' Set the background color to light-green.
cfRule2.Formatting.Fill.BackgroundColor = Color.FromArgb(255, &H9F, &HFB,
' Set the font color to blue-violet.
cfRule2.Formatting.Font.Color = Color.BlueViolet

```

The image below shows the result (the workbook is opened in Microsoft® Excel®). Cost values above the average in the first quarter are highlighted in yellow with a red font color and cost values one standard deviation below the average in the second quarter are highlighted in light-green with a blue-violet font color.

	C	D	
1	Cost (Qtr 1)	Cost (Qtr 2)	
2	\$5.44	\$5.80	
3	\$4.00	\$6.20	
4	\$9.38	\$9.30	
5	\$5.78	\$6.00	
6	\$3.50	\$3.20	
7	\$14.00	\$15.00	
8	\$10.50	\$10.00	
9	\$13.30	\$13.30	
10	\$9.00	\$9.85	
11	\$11.20	\$11.80	
12	\$12.70	\$13.45	
13	\$17.65	\$17.65	
14	\$10.00	\$9.50	
15	\$18.95	\$19.20	

How to: Format Cells that are Between or Not Between Two Values

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells that are Between or Not Between Two Values](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to apply the conditional formatting rule, highlighting cells that are between or not between two values.

1. To create a new conditional formatting rule represented by the RangeConditionalFormatting object, access the collection of conditional formats from the Worksheet.ConditionalFormattings property and call the ConditionalFormattingCollection.AddRangeConditionalFormatting method. Pass the following parameters:
 - A Range object that defines a range of cells to which the rule is applied.
 - A condition specified by one of the ConditionalFormattingRangeCondition enumeration values.
 - A string representing the low bound of the data subset inside or outside of which, formatted cells are located. Notice that the string can determine a formula to evaluate the lowest value.
 - A string representing the high bound of the data subset inside or outside of which, formatted cells are located. Notice that the string can determine a formula to evaluate the highest value.
2. Specify formatting options to be applied to cells if the condition is **true** using the ISupportsFormatting.Formatting property of the RangeConditionalFormatting object. Set the background and font colors.

To remove the RangeConditionalFormatting object, use the ConditionalFormattingCollection.Remove, ConditionalFormattingCollection.RemoveAt or ConditionalFormattingCollection.Clear methods.

C#

```
(ConditionalFormatting.cs)
// Create the rule to identify values below 7 and above 19 in cells F2 through F15.
RangeConditionalFormatting cfRule = worksheet.ConditionalFormattings.AddRangeConditionalFormatting(worksheet.Range["F2:F15"], ConditionalFormattingRangeCondition.btw, "7", "19");
// Specify formatting options to be applied to cells if the condition is true.
// Set the background color to yellow.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, 0xFA, 0xF7, 0xAA);
// Set the font color to red.
cfRule.Formatting.Font.Color = Color.Red;
```

Visual Basic

```
(ConditionalFormatting.vb)
' Create the rule to identify values below 7 and above 19 in cells F2 through F15.
Dim cfRule As RangeConditionalFormatting = worksheet.ConditionalFormattings.AddRangeConditionalFormatting(worksheet.Range["F2:F15"], ConditionalFormattingRangeCondition.btw, "7", "19")
' Specify formatting options to be applied to cells if the condition is true.
' Set the background color to yellow.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, &HFA, &HF7, &HAA)
' Set the font color to red.
cfRule.Formatting.Font.Color = Color.Red
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). Price values below \$7 and above \$19 are colored in yellow, and the font color is changed to red.

	F	
1	Price	
2	\$6.78	
3	\$7.90	
4	\$13.99	
5	\$7.99	
6	\$4.99	
7	\$16.99	
8	\$16.00	
9	\$16.49	
10	\$12.00	
11	\$16.00	
12	\$17.20	
13	\$19.25	
14	\$12.43	
15	\$22.99	
16		

How to: Format Top or Bottom Ranked Values

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Top or Bottom Ranked Values](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

- This example demonstrates how to apply a "top/bottom N" conditional formatting rule.
1. To create a new conditional formatting rule represented by the RankConditionalFormatting object, access the collection of conditional formats from the Worksheet.ConditionalFormattings property and call the ConditionalFormattingCollection.AddRankConditionalFormatting method. Pass the following parameters:
 - A Range object that defines a range of cells to which the rule is applied.
 - A condition specified by one of the ConditionalFormattingRankCondition enumeration values.
 - A positive integer value representing the number or percentage of the rank value.
 2. Specify formatting options to be applied to cells if the condition is **true**, using the ISupportsFormatting.Formatting property of the RankConditionalFormatting object. Define the background and font colors, and set the outline borders.

To remove the RankConditionalFormatting object, use the ConditionalFormattingCollection.Remove, ConditionalFormattingCollection.RemoveAt or ConditionalFormattingCollection.Clear methods.

C#

```
(ConditionalFormatting.cs)
// Create the rule to identify top three values in cells F2 through F15.
RankConditionalFormatting cfRule = worksheet.ConditionalFormattings.AddRankConditionalFormatting(worksheet
// Specify formatting options to be applied to cells if the condition is true.
// Set the background color to dark orchid.
cfRule.Formatting.Fill.BackgroundColor = Color.DarkOrchid;
// Set the outline borders.
cfRule.Formatting.Borders.SetOutsideBorders(Color.Black, BorderLineStyle.Thin);
// Set the font color to white.
cfRule.Formatting.Font.Color = Color.White;
```

Visual Basic

```
(ConditionalFormatting.vb)
' Create the rule to identify top three values in cells F2 through F15.
Dim cfRule As RankConditionalFormatting = worksheet.ConditionalFormattings
' Specify formatting options to be applied to cells if the condition is tr
' Set the background color to dark orchid.
cfRule.Formatting.Fill.BackgroundColor = Color.DarkOrchid
' Set the outline borders.
cfRule.Formatting.Borders.SetOutsideBorders(Color.Black, BorderLineStyle.T
' Set the font color to white.
cfRule.Formatting.Font.Color = Color.White
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). The top three price values are highlighted.

	F	
1	Price	
2	\$6.78	
3	\$7.90	
4	\$13.99	
5	\$7.99	
6	\$4.99	
7	\$16.99	
8	\$16.00	
9	\$16.49	
10	\$12.00	
11	\$16.00	
12	\$17.20	
13	\$19.25	
14	\$12.43	
15	\$22.99	
16		

How to: Format Cells based on the Text in the Cell

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells based on the Text in the Cell](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to create a rule that applies a conditional format based on the text in a cell.

1. To create a new conditional formatting rule represented by the TextConditionalFormatting object, access the collection of conditional formats from the Worksheet.ConditionalFormattings property and call the ConditionalFormattingCollection.AddTextConditionalFormatting method. Pass the following parameters:
 - A Range object that defines a range of cells to which the rule is applied.
 - A condition specified by one of the ConditionalFormattingTextCondition enumeration values.
 - A text string to be highlighted in a range of cells.
2. Specify formatting options to be applied to cells if the condition is true using the ISupportsFormatting.Formatting property of the TextConditionalFormatting object.

To remove the TextConditionalFormatting object, use the ConditionalFormattingCollection.Remove, ConditionalFormattingCollection.RemoveAt or ConditionalFormattingCollection.Clear methods.

C#

```
(ConditionalFormatting.cs)
// Create the rule to highlight values with the given text string in cells A2 through A15.
TextConditionalFormatting cfRule = worksheet.ConditionalFormattings.AddTextConditionalFormatting(worksheet
// Specify formatting options to be applied to cells if the condition is true.
// Set the background color to pink.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, 0xE1, 0x95, 0xC2);
```

Visual Basic

```
(ConditionalFormatting.vb)
' Create the rule to highlight values with the given text string in cells
Dim cfRule As TextConditionalFormatting = worksheet.ConditionalFormattings
' Specify formatting options to be applied to cells if the condition is tr
' Set the background color to pink.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, &HE1, &H95, &
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). In the list of authors, the name "Ray Bradbury" is highlighted in pink.

	A	
1	Author	
2	Ray Bradbury	
3	Ray Bradbury	
4	Ray Bradbury	
5	Ray Bradbury	
6	F. Scott Fitzgerald	
7	F. Scott Fitzgerald	
8	F. Scott Fitzgerald	
9	Harper Lee	
10	Ernest Hemingway	
11	Ernest Hemingway	
12	J.D. Salinger	
13	Gene Wolfe	
14	Gene Wolfe	
15	Gene Wolfe	
16		

How to: Format Unique or Duplicate Values, Blank Cells and Formula Errors

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Unique or Duplicate Values, Blank Cells and Formula Errors](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to specify the rule that highlights unique or duplicate values, formula errors, etc.

1. To create a new conditional formatting rule represented by the `SpecialConditionalFormatting` object, access the collection of conditional formats from the `Worksheet.ConditionalFormattings` property and call the `ConditionalFormattingCollection.AddSpecialConditionalFormatting` method. Pass the following parameters:
 - A `Range` object that defines a range of cells to which the rule is applied.
 - A condition specified by one of the `ConditionalFormattingSpecialCondition` enumeration values.
2. Specify formatting options to be applied to cells if the condition is **true**, using the `ISupportsFormatting.Formatting` property of the `SpecialConditionalFormatting` object.

To remove the `SpecialConditionalFormatting` object, use the `ConditionalFormattingCollection.Remove`, `ConditionalFormattingCollection.RemoveAt` or `ConditionalFormattingCollection.Clear` methods.

C#

```
(ConditionalFormatting.cs)
// Create the rule to identify unique values in cells A2 through A15.
SpecialConditionalFormatting cfRule = worksheet.ConditionalFormattings.AddSpecialConditionalFormatting(wor
// Specify formatting options to be applied to cells if the condition is true.
// Set the background color to yellow.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, 0xFA, 0xF7, 0xAA);
```

Visual Basic

```
(ConditionalFormatting.vb)
' Create the rule to identify unique values in cells A2 through A15.
Dim cfRule As SpecialConditionalFormatting = worksheet.ConditionalFormatti
' Specify formatting options to be applied to cells if the condition is tr
' Set the background color to yellow.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, &HFA, &HF7, &
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). Unique values in the list of authors are highlighted in yellow.

	A	
1	Author	
2	Ray Bradbury	
3	Ray Bradbury	
4	Ray Bradbury	
5	Ray Bradbury	
6	F. Scott Fitzgerald	
7	F. Scott Fitzgerald	
8	F. Scott Fitzgerald	
9	Harper Lee	
10	Ernest Hemingway	
11	Ernest Hemingway	
12	J.D. Salinger	
13	Gene Wolfe	
14	Gene Wolfe	
15	Gene Wolfe	
16		

How to: Format Cells that Contain Dates

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells that Contain Dates](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to apply a "date occurring..." conditional formatting rule.

1. To create a new conditional formatting rule represented by the `TimePeriodConditionalFormatting` object, access the collection of conditional formats from the `Worksheet.ConditionalFormattings` property and call the `ConditionalFormattingCollection.AddTimePeriodConditionalFormatting` method. Pass the following parameters:
 - A `Range` object that defines a range of cells to which the rule is applied.
 - A time period to be highlighted. This parameter is specified by one of the `ConditionalFormattingTimePeriod` enumeration values.
2. Specify formatting options to be applied to cells if the condition is true using the `ISupportsFormatting.Formatting` property of the `TimePeriodConditionalFormatting` object.

To remove the `TimePeriodConditionalFormatting` object, use the `ConditionalFormattingCollection.Remove`, `ConditionalFormattingCollection.RemoveAt` or `ConditionalFormattingCollection.Clear` methods.

C#

```
(ConditionalFormatting.cs)
// Create the rule to highlight today's dates in cells B2 through B6.
TimePeriodConditionalFormatting cfRule =
worksheet.ConditionalFormattings.AddTimePeriodConditionalFormatting(worksheet["$B$2:$B$6"], ConditionalFo
// Specify formatting options to be applied to cells if the condition is true.
// Set the background color to pink.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, 0xF2, 0xAE, 0xE3);
```

Visual Basic

```
(ConditionalFormatting.vb)
' Create the rule to highlight today's dates in cells B2 through B6.
Dim cfRule As TimePeriodConditionalFormatting = worksheet.ConditionalForma
' Specify formatting options to be applied to cells if the condition is tr
' Set the background color to pink.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, &HF2, &HAE, &
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). Today's date in the DUE DATE column is highlighted in pink.

	A	B	
1	TO DO	DUE DATE	
2	Visit the DevCon 2015	October 22, 2015	
3	Read The Art of Computer Programming till the end	December 31, 2015	
4	Buy the DevExpress Spreadsheet	November 12, 2013	
5	Download VS 2017	November 25, 2017	
6	Open a portal to another Universe	July 31, 2020	
7			

How to: Format Cells that are Less than, Greater than or Equal to a Value

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells that are Less than, Greater than or Equal to a Value](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to create the rule that uses a relational operator as a formatting criteria.

1. To create a new conditional formatting rule represented by the ExpressionConditionalFormatting object, access the collection of conditional formats from the Worksheet.ConditionalFormattings property and call the ConditionalFormattingCollection.AddExpressionConditionalFormatting method. Pass the following parameters:
 - A Range object that defines a range of cells to which the rule is applied.
 - A condition specified by one of the ConditionalFormattingExpressionCondition enumeration values.
 - A string that specifies the threshold value. Notice that the string can determine a formula to evaluate the threshold.
2. Specify formatting options to be applied to cells if the condition is **true**, using the ISupportsFormatting.Formatting property of the ExpressionConditionalFormatting object. Set the background and font colors.

To remove the ExpressionConditionalFormatting object, use the ConditionalFormattingCollection.Remove, ConditionalFormattingCollection.RemoveAt or ConditionalFormattingCollection.Clear methods.

C#

```
(ConditionalFormatting.cs)
// Create the rule to identify values that are above the average in cells F2 through F15.
ExpressionConditionalFormatting cfRule =
worksheet.ConditionalFormattings.AddExpressionConditionalFormatting(worksheet["$F$2:$F$15"], ConditionalFo
// Specify formatting options to be applied to cells if the condition is true.
// Set the background color to yellow.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, 0xFA, 0xF7, 0xAA);
// Set the font color to red.
cfRule.Formatting.Font.Color = Color.Red;
```

Visual Basic

```
(ConditionalFormatting.vb)
' Create the rule to identify values that are above the average in cells F
Dim cfRule As ExpressionConditionalFormatting = worksheet.ConditionalForma
' Specify formatting options to be applied to cells if the condition is tr
' Set the background color to yellow.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, &HFA, &HF7, &
' Set the font color to red.
cfRule.Formatting.Font.Color = Color.Red
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). Price values above the average are highlighted in yellow with red font.

	F	
1	Price	
2	\$6.78	
3	\$7.90	
4	\$13.99	
5	\$7.99	
6	\$4.99	
7	\$16.99	
8	\$16.00	
9	\$16.49	
10	\$12.00	
11	\$16.00	
12	\$17.20	
13	\$19.25	
14	\$12.43	
15	\$22.99	
16		

How to: Use a Formula to Determine which Cells to Format

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Use a Formula to Determine which Cells to Format](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to create the rule that uses a formula as a criterion to apply a conditional format.

1. To create a new conditional formatting rule represented by the `FormulaExpressionConditionalFormatting` object, access the collection of conditional formats from the `Worksheet.ConditionalFormattings` property and call the `ConditionalFormattingCollection.AddFormulaExpressionConditionalFormatting` method. Pass the following parameters:
 - A `Range` object that defines a range of cells to which the rule is applied.
 - A string value that determines a formula to evaluate.
2. Specify formatting options to be applied to cells if the condition is **true**, using the `ISupportsFormatting.Formatting` property of the `FormulaExpressionConditionalFormatting` object.

To remove the `FormulaExpressionConditionalFormatting` object, use the `ConditionalFormattingCollection.Remove`, `ConditionalFormattingCollection.RemoveAt` or `ConditionalFormattingCollection.Clear` methods.

C#

```
(ConditionalFormatting.cs)
// Create the rule to shade alternate rows without applying a new style.
FormulaExpressionConditionalFormatting cfRule = worksheet.ConditionalFormattings.AddFormulaExpressionConditionalFormatting(worksheet.Cells["A:A"], "1<2", true);
// Specify formatting options to be applied to cells if the condition is true.
// Set the background color to light blue.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, 0xBC, 0xDA, 0xF7);
```

Visual Basic

```
(ConditionalFormatting.vb)
' Create the rule to shade alternate rows without applying a new style.
Dim cfRule As FormulaExpressionConditionalFormatting = worksheet.ConditionalFormattings.AddFormulaExpressionConditionalFormatting(worksheet.Cells["A:A"], "1<2", true)
' Specify formatting options to be applied to cells if the condition is true.
' Set the background color to light blue.
cfRule.Formatting.Fill.BackgroundColor = Color.FromArgb(255, &HBC, &HDA, &HFF)
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). Alternate rows are shaded in light blue without applying a new style.

	A	B	C	D	E	F	G
1	Author	Title	Cost (Qtr 1)	Cost (Qtr 2)	Cost Trend	Price	Markup
2	Ray Bradbury	Dandelion Wine	\$5.44	\$5.80	\$0.36	\$6.78	17%
3	Ray Bradbury	Fahrenheit 451	\$4.00	\$6.20	\$2.20	\$7.90	27%
4	Ray Bradbury	A Sound of Thunder	\$9.38	\$9.30	(\$0.08)	\$13.99	50%
5	Ray Bradbury	The Martian Chronicles	\$5.78	\$6.00	\$0.22	\$7.99	33%
6	F. Scott Fitzgerald	The Beautiful and Damned	\$3.50	\$3.20	(\$0.30)	\$4.99	56%
7	F. Scott Fitzgerald	The Great Gatsby	\$14.00	\$15.00	\$1.00	\$16.99	13%
8	F. Scott Fitzgerald	Tender Is the Night	\$10.50	\$10.00	(\$0.50)	\$16.00	60%
9	Harper Lee	To Kill a Mockingbird	\$13.30	\$13.30	\$0.00	\$16.49	24%
10	Ernest Hemingway	The Old Man and the Sea	\$9.00	\$9.85	\$0.85	\$12.00	22%
11	Ernest Hemingway	A Firewell to Arms	\$11.20	\$11.80	\$0.60	\$16.00	36%
12	J.D. Salinger	The Catcher in the Rye	\$12.70	\$13.45	\$0.75	\$17.20	28%
13	Gene Wolfe	Nightside the Long Sun	\$17.65	\$17.65	\$0.00	\$19.25	9%
14	Gene Wolfe	Lake of the Long Sun	\$10.00	\$9.50	(\$0.50)	\$12.43	31%
15	Gene Wolfe	Exodus From the Long Sun	\$18.95	\$19.20	\$0.25	\$22.99	20%
16							
17							

How to: Format Cells Using a Two-Color Scale

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells Using a Two-Color Scale](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to apply a two-color scale conditional formatting rule.

1. First of all, specify the minimum and maximum thresholds of a range using the ConditionalFormattingCollection.CreateValue method that creates an instance of the ConditionalFormattingValue object. This object provides access to threshold values and their types. The type of the threshold value is determined by one of the ConditionalFormattingValueType enumeration values and can be a number, percent, formula, or percentile. Call the ConditionalFormattingCollection.CreateValue method with the ConditionalFormattingValueType.MinMax parameter to set the minimum and maximum thresholds to the lowest and highest values in a range of cells, respectively.
2. To apply a conditional formatting rule represented by the ColorScale2ConditionalFormatting object, access the collection of conditional formats from the Worksheet.ConditionalFormattings property and call the ConditionalFormattingCollection.AddColorScale2ConditionalFormatting method with the following parameters:
 - A Range object that defines a range of cells to which the rule is applied.
 - A minimum threshold specified by the ConditionalFormattingValue object.
 - A color corresponding to the minimum value in a range of cells.
 - A maximum threshold specified by the ConditionalFormattingValue object.
 - A color corresponding to the maximum value in a range of cells.

To remove the ColorScale2ConditionalFormatting object, use the ConditionalFormattingCollection.Remove, ConditionalFormattingCollection.RemoveAt or ConditionalFormattingCollection.Clear methods.

C#

```
(ConditionalFormatting.cs)
ConditionalFormattingCollection conditionalFormattings = worksheet.ConditionalFormattings;
// Set the minimum threshold to the lowest value in the range of cells.
ConditionalFormattingValue minPoint = conditionalFormattings.CreateValue(ConditionalFormattingValueType.MinMax, 0);
// Set the maximum threshold to the highest value in the range of cells.
ConditionalFormattingValue maxPoint = conditionalFormattings.CreateValue(ConditionalFormattingValueType.MinMax, 100);
// Create the two-color scale rule to differentiate low and high values in cells C2 through D15. Blue represents the minimum value and yellow represents the maximum value.
ColorScale2ConditionalFormatting cfRule = conditionalFormattings.AddColorScale2ConditionalFormatting(worksheet.Range["C2:D15"], minPoint, maxPoint, Color.Blue, Color.Yellow);
```

Visual Basic

```
(ConditionalFormatting.vb)
Dim conditionalFormattings As ConditionalFormattingCollection = worksheet.ConditionalFormattings
' Set the minimum threshold to the lowest value in the range of cells.
Dim minPoint As ConditionalFormattingValue = conditionalFormattings.CreateValue(ConditionalFormattingValueType.MinMax, 0)
' Set the maximum threshold to the highest value in the range of cells.
Dim maxPoint As ConditionalFormattingValue = conditionalFormattings.CreateValue(ConditionalFormattingValueType.MinMax, 100)
' Create the two-color scale rule to differentiate low and high values in cells C2 through D15. Blue represents the minimum value and yellow represents the maximum value.
Dim cfRule As ColorScale2ConditionalFormatting = conditionalFormattings.AddColorScale2ConditionalFormatting(worksheet.Range["C2:D15"], minPoint, maxPoint, Color.Blue, Color.Yellow)
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). Cost distribution is shown using a gradation of two colors. Blue represents the lower values and yellow represents the higher values.

	C	D	
1	Cost (Qtr 1)	Cost (Qtr 2)	
2	\$5.44	\$5.80	
3	\$4.00	\$6.20	
4	\$9.38	\$9.30	
5	\$5.78	\$6.00	
6	\$3.50	\$3.20	
7	\$14.00	\$15.00	
8	\$10.50	\$10.00	
9	\$13.30	\$13.30	
10	\$9.00	\$9.85	
11	\$11.20	\$11.80	
12	\$12.70	\$13.45	
13	\$17.65	\$17.65	
14	\$10.00	\$9.50	
15	\$18.95	\$19.20	
16			

How to: Format Cells Using a Three-Color Scale

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells Using a Three-Color Scale](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to apply a three-color scale conditional formatting rule.

1. First of all, specify the minimum, midpoint and maximum thresholds of a range to which the rule will be applied. Threshold values are determined by the ConditionalFormattingValue object that can be accessed via the ConditionalFormattingCollection.CreateValue method. The type of the threshold value is specified by one of the ConditionalFormattingValueType enumeration values and can be a number, percent, formula, or percentile. Call the ConditionalFormattingCollection.CreateValue method with the ConditionalFormattingValueType.MinMax parameter to set the minimum and maximum thresholds to the lowest and highest values in a range of cells, respectively.
2. To apply a conditional formatting rule represented by the ColorScale3ConditionalFormatting object, access the collection of conditional formats from the Worksheet.ConditionalFormattings property and call the ConditionalFormattingCollection.AddColorScale3ConditionalFormatting method with the following parameters:
 - A Range object that defines a range of cells to which the rule is applied.
 - A minimum threshold specified by the ConditionalFormattingValue object.
 - A color corresponding to the minimum value in a range of cells.
 - A midpoint threshold specified by the ConditionalFormattingValue object.
 - A color corresponding to the middle value in a range of cells.
 - A maximum threshold specified by the ConditionalFormattingValue object.
 - A color corresponding to the maximum value in a range of cells.

To remove the ColorScale3ConditionalFormatting object, use the ConditionalFormattingCollection.Remove, ConditionalFormattingCollection.RemoveAt or ConditionalFormattingCollection.Clear methods.

C#

```
(ConditionalFormatting.cs)
ConditionalFormattingCollection conditionalFormattings = worksheet.ConditionalFormattings;
// Set the minimum threshold to the lowest value in the range of cells using the MIN() formula.
ConditionalFormattingValue minPoint = conditionalFormattings.CreateValue(ConditionalFormattingValueType.Fo
// Set the midpoint threshold to the 50th percentile.
ConditionalFormattingValue midPoint = conditionalFormattings.CreateValue(ConditionalFormattingValueType.Pe
// Set the maximum threshold to the highest value in the range of cells using the MAX() formula.
ConditionalFormattingValue maxPoint = conditionalFormattings.CreateValue(ConditionalFormattingValueType.Nu
// Create the three-color scale rule to determine how values in cells C2 through D15 vary. Red represents
ColorScale3ConditionalFormatting cfRule = conditionalFormattings.AddColorScale3ConditionalFormatting(work
```

Visual Basic

```
(ConditionalFormatting.vb)
Dim conditionalFormattings As ConditionalFormattingCollection = worksheet.
' Set the minimum threshold to the lowest value in the range of cells usin
Dim minPoint As ConditionalFormattingValue = conditionalFormattings.Create
' Set the midpoint threshold to the 50th percentile.
Dim midPoint As ConditionalFormattingValue = conditionalFormattings.Create
' Set the maximum threshold to the highest value in the range of cells usi
Dim maxPoint As ConditionalFormattingValue = conditionalFormattings.Create
' Create the three-color scale rule to determine how values in cells C2 th
Dim cfRule As ColorScale3ConditionalFormatting = conditionalFormattings.Ad
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). Cost distribution is shown using a gradation of three colors. Red represents the lower values, yellow represents the medium values and sky blue represents the higher values.

	C	D	
1	Cost (Qtr 1)	Cost (Qtr 2)	
2	\$5.44	\$5.80	
3	\$4.00	\$6.20	
4	\$9.38	\$9.30	
5	\$5.78	\$6.00	
6	\$3.50	\$3.20	
7	\$14.00	\$15.00	
8	\$10.50	\$10.00	
9	\$13.30	\$13.30	
10	\$9.00	\$9.85	
11	\$11.20	\$11.80	
12	\$12.70	\$13.45	
13	\$17.65	\$17.65	
14	\$10.00	\$9.50	
15	\$18.95	\$19.20	
16			

How to: Format Cells Using Data Bars

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells Using Data Bars](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to apply a data bar conditional formatting rule.

1. First of all, specify the minimum and maximum thresholds corresponding to the shortest and longest bars, respectively. To do this, use the `ConditionalFormattingCollection.CreateValue` method, which creates an instance of the `ConditionalFormattingValue` object. This object provides access to threshold values and their types. The type of the threshold value is determined by one of the `ConditionalFormattingValueType` enumeration values and can be a number, percent, formula, or percentile. To automatically set the minimum (maximum) threshold to the lowest (highest) value in a range to which the rule will be applied, use the `ConditionalFormattingCollection.CreateValue` method with the `ConditionalFormattingValueType.MinMax` enumeration value passed as a parameter. To compare values in a range of cells based on their distance from zero, use the `ConditionalFormattingCollection.CreateValue` method with the `ConditionalFormattingValueType.Auto` parameter.
2. To apply a conditional formatting rule represented by the `DataBarConditionalFormatting` object, access the collection of conditional formats from the `Worksheet.ConditionalFormattings` property and call the `ConditionalFormattingCollection.AddDataBarConditionalFormatting` method with the following parameters: the range of cells to which the rule is applied, the minimum and maximum thresholds, and the bar color.
3. Modify the behavior of bars: set the border color using the `DataBarConditionalFormatting.BorderColor` property and specify whether or not to use the gradient fill type by setting the `DataBarConditionalFormatting.GradientFill` property. If this property is set to **false**, the solid fill type is applied. To change the direction of bars, use the `DataBarConditionalFormatting.Direction` property.
4. If cells contain **negative** values, provide settings for the negative bar. Set its fill color from the `DataBarConditionalFormatting.NegativeBarColor` property and specify the border line color using the `DataBarConditionalFormatting.NegativeBarBorderColor` property.
5. Provide settings for the axis separating positive and negative bars. Specify the axis position using the `DataBarConditionalFormatting.AxisPosition` property. Set the axis color from the `DataBarConditionalFormatting.AxisColor` property.
6. Specify whether to show or hide values of the cells where the rule is applied by setting the `DataBarConditionalFormatting.ShowValue` property.

To remove the `DataBarConditionalFormatting` object, use the `ConditionalFormattingCollection.Remove`, `ConditionalFormattingCollection.RemoveAt` or `ConditionalFormattingCollection.Clear` methods.

C#

```
(ConditionalFormatting.cs)
ConditionalFormattingCollection conditionalFormattings = worksheet.ConditionalFormattings;
// Set the value corresponding to the shortest bar to the lowest value.
ConditionalFormattingValue lowBound1 = conditionalFormattings.CreateValue(ConditionalFormattingValueType.Min);
// Set the value corresponding to the longest bar to the highest value.
ConditionalFormattingValue highBound1 = conditionalFormattings.CreateValue(ConditionalFormattingValueType.Max);
// Create the rule to compare values in cells E2 through E15 using data bars.
DataBarConditionalFormatting cfRule1 = conditionalFormattings.AddDataBarConditionalFormatting(worksheet.Range("E2:E15"));
// Set the positive bar border color to green.
cfRule1.BorderColor = DXColor.Green;
// Set the negative bar color to red.
cfRule1.NegativeBarColor = DXColor.Red;
// Set the negative bar border color to red.
cfRule1.NegativeBarBorderColor = DXColor.Red;
// Set the axis position to display the axis in the middle of the cell.
cfRule1.AxisPosition = ConditionalFormattingDataBarAxisPosition.Middle;
// Set the axis color to dark blue.
cfRule1.AxisColor = Color.DarkBlue;
// Set the value corresponding to the shortest bar to 0 percent.
ConditionalFormattingValue lowBound2 = conditionalFormattings.CreateValue(ConditionalFormattingValueType.Min);
// Set the value corresponding to the longest bar to 100 percent.
ConditionalFormattingValue highBound2 = conditionalFormattings.CreateValue(ConditionalFormattingValueType.Max);
// Create the rule to compare values in cells G2 through G15 using data bars.
DataBarConditionalFormatting cfRule2 = conditionalFormattings.AddDataBarConditionalFormatting(worksheet.Range("G2:G15"));
// Set the data bar border color to sky blue.
cfRule2.BorderColor = DXColor.SkyBlue;
// Specify the solid fill type.
cfRule2.GradientFill = false;
// Hide values of cells to which the rule is applied.
cfRule2.ShowValue = false;
```

Visual Basic	
---------------------	--

```
(ConditionalFormatting.vb)
Dim conditionalFormattings As ConditionalFormattingCollection = worksheet.
' Set the value corresponding to the shortest bar to the lowest value.
Dim lowBound1 As ConditionalFormattingValue = conditionalFormattings.Creat
' Set the value corresponding to the longest bar to the highest value.
Dim highBound1 As ConditionalFormattingValue = conditionalFormattings.Crea
' Create the rule to compare values in cells E2 through E15 using data bar
Dim cfRule1 As DataBarConditionalFormatting = conditionalFormattings.AddDa
' Set the positive bar border color to green.
cfRule1.BorderColor = DXColor.Green
' Set the negative bar color to red.
cfRule1.NegativeBarColor = DXColor.Red
' Set the negative bar border color to red.
cfRule1.NegativeBarBorderColor = DXColor.Red
' Set the axis position to display the axis in the middle of the cell.
cfRule1.AxisPosition = ConditionalFormattingDataBarAxisPosition.Middle
' Set the axis color to dark blue.
cfRule1.AxisColor = Color.DarkBlue
' Set the value corresponding to the shortest bar to 0 percent.
Dim lowBound2 As ConditionalFormattingValue = conditionalFormattings.Creat
' Set the value corresponding to the longest bar to 100 percent.
Dim highBound2 As ConditionalFormattingValue = conditionalFormattings.Crea
' Create the rule to compare values in cells G2 through G15 using data bar
Dim cfRule2 As DataBarConditionalFormatting = conditionalFormattings.AddDa
' Set the data bar border color to sky blue.
cfRule2.BorderColor = DXColor.SkyBlue
' Specify the solid fill type.
cfRule2.GradientFill = False
' Hide values of cells to which the rule is applied.
cfRule2.ShowValue = False
```

The image below shows the result (the workbook is opened in Microsoft® Excel®). The data bars allow you to compare values in the "Cost Trend" and "Markup" columns.

	E	F	G	
1	Cost Trend	Price	Markup	
2	\$0.36	\$6.78		
3	\$2.20	\$7.90		
4	(\$0.08)	\$13.99		
5	\$0.22	\$7.99		
6	(\$0.30)	\$4.99		
7	\$1.00	\$16.99		
8	(\$0.50)	\$16.00		
9	\$0.00	\$16.49		
10	\$0.35	\$12.00		
11	\$0.50	\$16.00		
12	\$0.75	\$17.20		
13	\$0.00	\$19.25		
14	(\$0.50)	\$12.43		
15	\$0.25	\$22.99		
16				

How to: Format Cells Using Icon Sets

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells Using Icon Sets](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4959>.

This example demonstrates how to apply an icon set conditional formatting rule.

1. First of all, specify threshold values separating data categories, each of which will be represented by its own icon from the icon set. To do this, use the `ConditionalFormattingCollection.CreateIconSetValue` method that creates an instance of the `ConditionalFormattingIconSetValue` object. This object inherits from the `ConditionalFormattingValue` base interface and provides access to threshold values and their types. The type of the threshold value is determined by one of the `ConditionalFormattingValueType` enumeration values and can be a number, percent, formula, or percentile. The third parameter in the `ConditionalFormattingCollection.CreateIconSetValue` method defines a relational operator specified by one of the `ConditionalFormattingValueOperator` enumeration values.
2. To apply a conditional formatting rule represented by the `IconSetConditionalFormatting` object, access the collection of conditional formats from the `Worksheet.ConditionalFormattings` property and call the `ConditionalFormattingCollection.AddIconSetConditionalFormatting` method. Pass the following parameters:
 - A `Range` object that defines a range of cells to which the rule is applied.
 - An icon set to be displayed, determined by one of the `IconSetType` enumeration values.
 - An array of threshold values specified by the `ConditionalFormattingIconSetValue` object.
3. Specify whether to show or hide the values of cells to which the conditional formatting rule is applied using the `IconSetConditionalFormatting.ShowValue` property.

To specify the custom icon style for the created rule, do the following:

1. Set the `IconSetConditionalFormatting.IsCustom` property to **true** to indicate that the custom icon style is used.
2. Initialize an instance of the `ConditionalFormattingCustomIcon` structure using the default constructor. Use the `ConditionalFormattingCustomIcon.IconSet` and `ConditionalFormattingCustomIcon.IconIndex` properties of the `ConditionalFormattingCustomIcon` object to specify the icon set from which you wish to get the icon and the index of the desired icon in the set, respectively.
3. Set the custom icon via the `IconSetConditionalFormatting.SetCustomIcon` method with the following parameters: the position in the initial icon set at which the custom icon will be added, and the custom icon specified by the `ConditionalFormattingCustomIcon` object.

To remove the `IconSetConditionalFormatting` object, use the `ConditionalFormattingCollection.Remove`, `ConditionalFormattingCollection.RemoveAt` or `ConditionalFormattingCollection.Clear` methods.

C#

```

(ConditionalFormatting.cs)
ConditionalFormattingCollection conditionalFormattings = worksheet.ConditionalFormattings;
// Set the first threshold to the lowest value in the range of cells using the MIN() formula.
ConditionalFormattingIconSetValue minPoint = conditionalFormattings.CreateIconSetValue(ConditionalFormatt
// Set the second threshold to 0.
ConditionalFormattingIconSetValue midPoint = conditionalFormattings.CreateIconSetValue(ConditionalFormatt
// Set the third threshold to 0.01.
ConditionalFormattingIconSetValue maxPoint = conditionalFormattings.CreateIconSetValue(ConditionalFormatt
// Create the rule to apply a specific icon from the three arrow icon set to each cell in the range E2:E
IconSetConditionalFormatting cfRule = conditionalFormattings.AddIconSetConditionalFormatting(worksheet.Ran
// Specify the custom icon to be displayed if the second condition is true.
// To do this, set the IconSetConditionalFormatting.IsCustom property to true, which is false by default.
cfRule.IsCustom = true;
// Initialize the ConditionalFormattingCustomIcon object.
ConditionalFormattingCustomIcon cfCustomIcon = new ConditionalFormattingCustomIcon();
// Specify the icon set where you wish to get the icon.
cfCustomIcon.IconSet = IconSetType.TrafficLights13;
// Specify the index of the desired icon in the set.
cfCustomIcon.IconIndex = 1;
// Add the custom icon at the specified position in the initial icon set.
cfRule.SetCustomIcon(1, cfCustomIcon);
// Hide values of cells to which the rule is applied.
cfRule.ShowValue = false;

```

Visual Basic

```

(ConditionalFormatting.vb)
Dim conditionalFormattings As ConditionalFormattingCollection = worksheet.
' Set the first threshold to the lowest value in the range of cells using
Dim minPoint As ConditionalFormattingIconSetValue = conditionalFormattings
' Set the second threshold to 0.
Dim midPoint As ConditionalFormattingIconSetValue = conditionalFormattings
' Set the third threshold to 0.01.
Dim maxPoint As ConditionalFormattingIconSetValue = conditionalFormattings
' Create the rule to apply a specific icon from the three arrow icon set t
Dim cfRule As IconSetConditionalFormatting = conditionalFormattings.AddIconSetConditionalFormatting
' Specify the custom icon to be displayed if the second condition is true.
' To do this, set the IconSetConditionalFormatting.IsCustom property to true.
cfRule.IsCustom = True
' Initialize the ConditionalFormattingCustomIcon object.
Dim cfCustomIcon As New ConditionalFormattingCustomIcon()
' Specify the icon set where you wish to get the icon.
cfCustomIcon.IconSet = IconSetType.TrafficLights13
' Specify the index of the desired icon in the set.
cfCustomIcon.IconIndex = 1
' Add the custom icon at the specified position in the initial icon set.
cfRule.SetCustomIcon(1, cfCustomIcon)
' Hide values of cells to which the rule is applied.
cfRule.ShowValue = False

```

The image below shows the result (the workbook is opened in Microsoft® Excel®). The icons applied allow you to identify upward and downward cost trends.

	C	D	E	
1	Cost (Qtr 1)	Cost (Qtr 2)	Cost Trend	
2	\$5.44	\$5.80	↑	
3	\$4.00	\$6.20	↑	
4	\$9.38	\$9.30	↓	
5	\$5.78	\$6.00	↑	
6	\$3.50	\$3.20	↓	
7	\$14.00	\$15.00	↑	
8	\$10.50	\$10.00	↓	
9	\$13.30	\$13.30	●	
10	\$9.00	\$9.85	↑	
11	\$11.20	\$11.80	↑	
12	\$12.70	\$13.45	↑	
13	\$17.65	\$17.65	●	
14	\$10.00	\$9.50	↓	
15	\$18.95	\$19.20	↑	
16				

Group Data

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Grouping](#)

This section contains examples that demonstrate how to split a large amount of data into separate groups and display summary rows and columns for each group. Data grouping is useful when you wish to temporally hide unnecessary rows or columns to simplify navigation within large worksheets and display only significant information.

	A	B	C	D	E	F	G
1							
2		Arizona	Q1	Q2	Q3	Q4	Yearly total
3		HD Video Player	\$ 2,970.00	\$ 7,590.00	\$ 8,250.00	\$ 10,230.00	\$ 29,040.00
4		SuperLED 50	\$ 4,800.00	\$ 11,200.00	\$ 9,600.00	\$ 4,800.00	\$ 30,400.00
5		DesktopLED 21	\$ 4,025.00	\$ 1,050.00	\$ 2,800.00	\$ 2,275.00	\$ 10,150.00
6		Projector Plus HD	\$ 7,500.00	\$ 3,000.00	\$ 9,750.00	\$ 8,250.00	\$ 28,500.00
7		Quarterly total	\$ 19,295.00	\$ 22,840.00	\$ 30,400.00	\$ 25,555.00	\$ 98,090.00
8		California	Q1	Q2	Q3	Q4	Yearly total
13		Quarterly total	\$ 45,545.00	\$ 46,225.00	\$ 53,940.00	\$ 30,605.00	\$ 176,315.00
14		Grand Total	\$ 64,840.00	\$ 69,065.00	\$ 84,340.00	\$ 56,160.00	\$ 274,405.00
15							

- [How to: Outline Data Manually](#)
- [How to: Outline Data Automatically](#)
- [How to: Insert Subtotals in a Data Range](#)

How to: Outline Data Manually

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Grouping](#) > [How to: Outline Data Manually](#)

You can group rows and columns to make it easier to analyze a large amount of data, and simplify navigation within large worksheets. This functionality provides the capability to split your data into separate groups and hide unnecessary details in a worksheet.

Select the action you wish to perform.

- [Group and Ungroup Rows](#)
- [Group and Ungroup Columns](#)

Group and Ungroup Rows

- Group

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(GroupAndOutlineActions.cs)
Worksheet worksheet = workbook.Worksheets["Grouping"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Group four rows starting from the third row and collapse the group.
worksheet.Rows.Group(2, 5, true);
// Group four rows starting from the ninth row and expand the group.
worksheet.Rows.Group(8, 11, false);
// Create the outer group of rows by grouping rows 2 through 13.
worksheet.Rows.Group(1, 12, false);
```

The image below shows the result (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F	G
1							
2		Arizona	Q1	Q2	Q3	Q4	Yearly total
7		Quarterly total	\$ 19,295.00	\$ 22,840.00	\$ 30,400.00	\$ 25,555.00	\$ 98,090.00
8		California	Q1	Q2	Q3	Q4	Yearly total
9		HD Video Player	\$ 9,570.00	\$ 11,550.00	\$ 12,540.00	\$ 5,280.00	\$ 38,940.00
10		SuperLED 50	\$ 20,800.00	\$ 17,600.00	\$ 20,800.00	\$ 6,400.00	\$ 65,600.00
11		DesktopLED 21	\$ 5,425.00	\$ 5,075.00	\$ 5,600.00	\$ 5,425.00	\$ 21,525.00
12		Projector Plus HD	\$ 9,750.00	\$ 12,000.00	\$ 15,000.00	\$ 13,500.00	\$ 50,250.00
13		Quarterly total	\$ 45,545.00	\$ 46,225.00	\$ 53,940.00	\$ 30,605.00	\$ 176,315.00
14		Grand Total	\$ 64,840.00	\$ 69,065.00	\$ 84,340.00	\$ 56,160.00	\$ 274,405.00

Ungroup

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(GroupAndOutlineActions.cs)
Worksheet worksheet = workbook.Worksheets["Grouping and Outline"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Ungroup four rows (from the third row to the sixth row) and display collapsed data.
worksheet.Rows.UnGroup(2, 5, true);
// Ungroup four rows (from the ninth row to the twelfth row).
worksheet.Rows.UnGroup(8, 11, false);
// Remove the outer group of rows.
worksheet.Rows.UnGroup(1, 12, false);
```

Group and Ungroup Columns

- Group
- Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

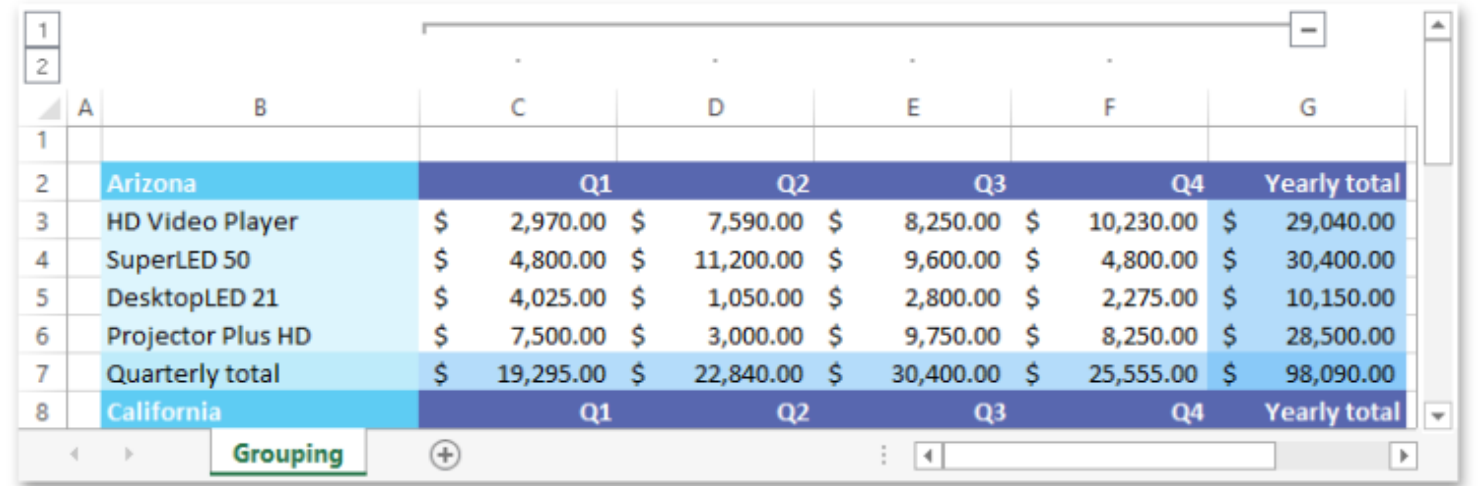
C#

(GroupAndOutlineActions.cs)
Worksheet worksheet = workbook.Worksheets["Grouping"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Group four columns starting from the third column "C" and expand the group
worksheet.Columns.Group(2, 5, false);

Visual Basic

(GroupAndOutlineActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Grouping")
workbook.Worksheets.ActiveWorksheet = worksheet
' Group four columns starting from the third column "C" and expand the group
worksheet.Columns.Group(2, 5, False)

The image below shows the result (the workbook is opened in Microsoft® Excel®).



- Ungroup
- Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T2176> 15.

C#	
<pre>(GroupAndOutlineActions.cs) Worksheet worksheet = workbook.Worksheets["Grouping and Outline"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Ungroup four columns (from the column "C" to the column "F"). worksheet.Columns.UnGroup(2, 5, false);</pre>	
Visual Basic	
<pre>(GroupAndOutlineActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("Grouping and Outline") workbook.Worksheets.ActiveWorksheet = worksheet ' Ungroup four columns (from the column "C" to the column "F"). worksheet.Columns.UnGroup(2, 5, False)</pre>	

See Also

- [How to: Outline Data Automatically](#)
- [How to: Insert Subtotals in a Data Range](#)

How to: Outline Data Automatically

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Grouping](#) > [How to: Outline Data Automatically](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

The example below demonstrates how to outline data automatically using the `Worksheet.AutoOutline` method.

The `Worksheet.AutoOutline` method uses worksheet formulas to determine how to group data. Thus, make sure that the worksheet you wish to outline contains summary rows or columns with formulas based on which the outline will be created. If you did not specify any summary formulas, the data will not be grouped.

To remove an outline for the specified worksheet, use the `Worksheet.ClearOutline` method.

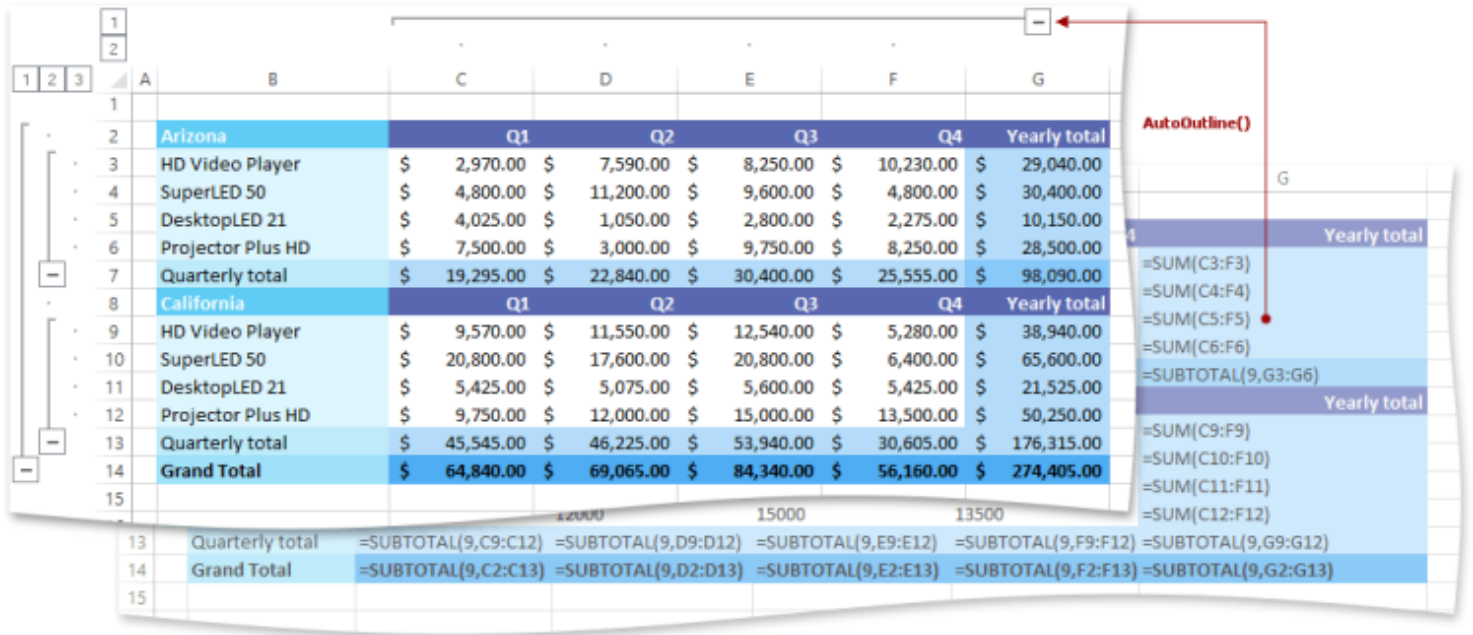
C#

(GroupAndOutlineActions.cs)
Worksheet worksheet = workbook.Worksheets["Grouping"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Outline data automatically based on the summary formulas.
worksheet.AutoOutline();

Visual Basic

(GroupAndOutlineActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Grouping")
workbook.Worksheets.ActiveWorksheet = worksheet
' Outline data automatically based on the summary formulas.
worksheet.AutoOutline()

The image below shows the source worksheet to which the `Worksheet.AutoOutline` method is applied, and the resulting document with the corresponding outlines of rows and columns (the workbook is opened in Microsoft® Excel®).



See Also

- [How to: Outline Data Manually](#)
- [How to: Insert Subtotals in a Data Range](#)

How to: Insert Subtotals in a Data Range

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Grouping](#) > [How to: Insert Subtotals in a Data Range](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

The example below demonstrates how to use the `Worksheet.Subtotal` method to automatically create outlines for a sorted range and summarize data in each group using the [SUBTOTAL function](#).

Important

Before subtotalling, make sure that the range you wish to subtotal has column headings in the first row and the data in this range is sorted by a column at each change in which a subtotal row will be inserted.

To insert subtotals, do the following.

1. Specify the Range object, which contains data you wish to subtotal
2. Create a list of column indexes that define columns for which the subtotals should be calculated. In this example, the SUBTOTAL function will be calculated only for column "D" (with Column.Index equal to 3).
3. Call the `Worksheet.Subtotal` method and pass the following parameters: the specified data range to be subtotaled, the index of the column by which you wish to group your data, the specified list of columns to which the subtotals should be added, the [code of the function](#) to be used in calculating subtotals and the text to be displayed in the summary rows.

In this example, a subtotal row will be inserted each time a value in column "B" (with Column.Index equal to 1) changes. The SUM function with code 9 is used to calculate subtotals.

C#

```
(GroupAndOutlineActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional Sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
Range dataRange = worksheet["B3:E23"];
// Specify that subtotals should be calculated for the column "D".
List<int> subtotalColumnsList = new List<int>();
subtotalColumnsList.Add(3);
// Insert subtotals by each change in the column "B" and calculate the SUM fuction for the related rows in
worksheet.Subtotal(dataRange, 1, subtotalColumnsList, 9, "Total");
```

Visual Basic

```
(GroupAndOutlineActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Regional Sales")
workbook.Worksheets.ActiveWorksheet = worksheet
Dim dataRange As Range = worksheet("B3:E23")
' Specify that subtotals should be calculated for the column "D".
Dim subtotalColumnsList As New List(Of Integer)()
subtotalColumnsList.Add(3)
' Insert subtotals by each change in the column "B" and calculate the SUM
worksheet.Subtotal(dataRange, 1, subtotalColumnsList, 9, "Total")
```

The image below shows the result (the workbook is opened in Microsoft® Excel®).

1	2	3	A	B	C	D
			1			
			2	Region	Product	Sales
		•	3	East	Camembert Pierrot	\$ 6,750.00
		•	4	East	Gorgonzola Telino	\$ 4,500.00
		•	5	East	Mozzarella di Giovanni	\$ 5,200.00
		•	6	East	Mozzarella di Giovanni	\$ 3,800.00
		-	7	East Total		20250
		•	8	West	Gorgonzola Telino	\$ 12,500.00
		•	9	West	Mascarpone Fabioli	\$ 8,600.00
		•	10	West	Mozzarella di Giovanni	\$ 6,580.00
		-	11	West Total		27680
		-	12	Grand Total		47930
			13			

See Also[How to: Outline Data Manually](#)[How to: Outline Data Automatically](#)

Filter Data

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#)

This section contains examples that demonstrate how to use the AutoFilter functionality to arrange large amounts of data by displaying only rows that meet the filtering criteria.

- [How to: Enable Filtering](#)
- [How to: Filter by Cell Values](#)
- [How to: Filter by Date Values](#)
- [How to: Apply a Custom Date Filter](#)
- [How to: Apply a Custom Text Filter](#)
- [How to: Apply a Custom Number Filter](#)
- [How to: Apply a Dynamic Filter](#)
- [How to: Filter Top or Bottom Ranked Values](#)
- [How to: Sort Data in the Filtered Range](#)
- [How to: Reapply a Filter](#)
- [How to: Clear a Filter](#)

How to: Enable Filtering

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Enable Filtering](#)

The examples below demonstrate how to enable a filtering functionality in a workbook.

- [Filter a Worksheet Range](#)
- [Filter a Table](#)

Filter a Worksheet Range

To activate filtering for a specific range in a worksheet, do the following.

1. Use the Worksheet.AutoFilter property to get access to the SheetAutoFilter object. This object inherits the AutoFilterBase interface, which provides basic methods and properties used to apply, clear or disable a filter and sort values in the filtered range.
2. Call the SheetAutoFilter.Apply method to enable filtering. Pass the Range object you wish to filter as a parameter.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

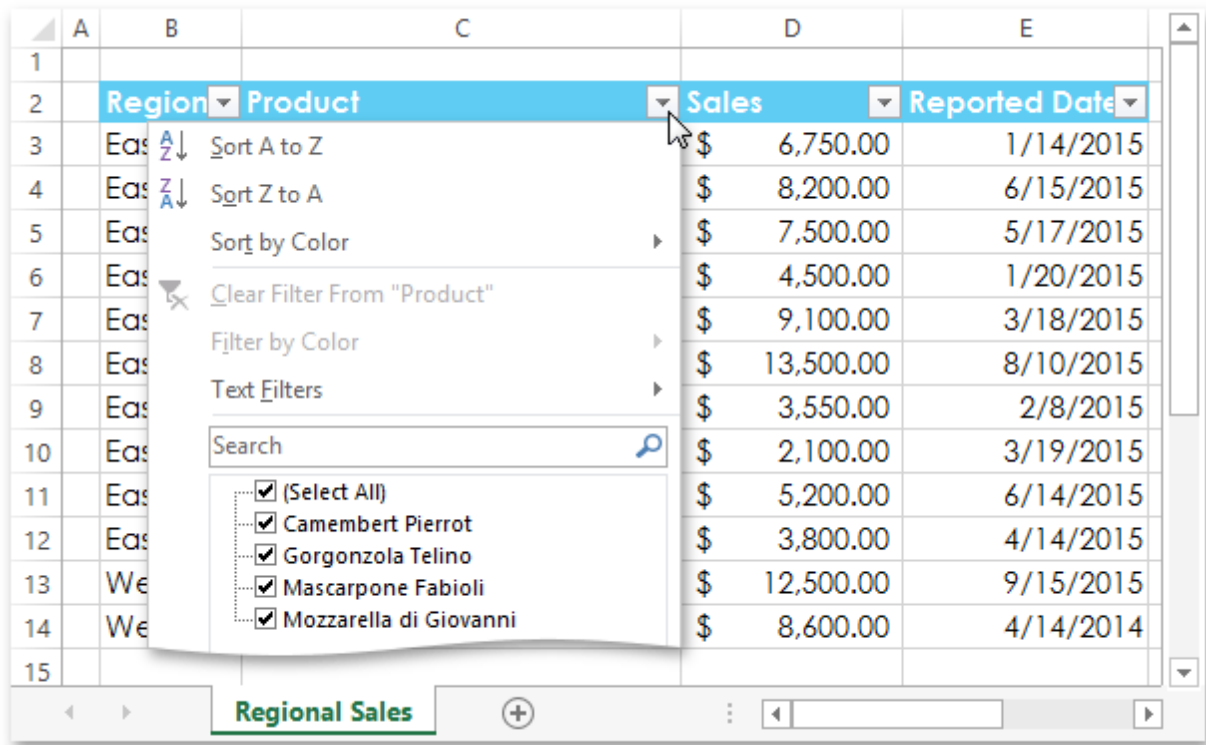
C#

```
(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
```

Visual Basic

```
(AutoFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Regional sales")
workbook.Worksheets.ActiveWorksheet = worksheet
' Enable filtering for the specified cell range.
Dim range As Range = worksheet("B2:E23")
worksheet.AutoFilter.Apply(range)
```

The image below illustrates the result (the workbook is opened in Microsoft® Excel®). Once filtering is activated, a drop-down arrow appears on the right side of each column header in the filtered range. A user can click the arrow of the desired column and select the required filter options in the AutoFilter drop-down menu.



Filter a Table

By default, when you [create a table](#) in a worksheet, it already has the AutoFilter functionality turned on. Thus, to filter data in the table columns, get access to the required table in the worksheet's TableCollection, and then use the Table.AutoFilter property to return the TableAutoFilter object. This object inherits the AutoFilterBase base interface, which provides common methods and properties used to apply, clear or disable a table filter and sort values in the table columns in the same manner as it can be done for a worksheet range.

C#
<pre>// Access a table. Table table = worksheet.Tables[0]; // Filter values in the "Amount" column that are greater than 75\$. table.AutoFilter.Columns[4].ApplyCustomFilter(75, FilterComparisonOperator</pre>
Visual Basic
<pre>' Access a table. Dim table As Table = worksheet.Tables(0) ' Filter values in the "Amount" column that are greater than 75\$. table.AutoFilter.Columns(4).ApplyCustomFilter(75, FilterComparisonOperator</pre>

The image below illustrates the result of executing the code (the workbook is opened in Microsoft® Excel®).

The screenshot shows a spreadsheet with a table containing columns: Product, Price, Quantity, Discount, and Amount. The 'Amount' column header is selected, and a context menu is open. The menu includes options for sorting (Sort Smallest to Largest, Sort Largest to Smallest, Sort by Color) and filtering (Clear Filter From "Amount", Filter by Color, Number Filters). The 'Number Filters' option is selected, opening a submenu with various comparison operators. The 'Greater Than Or Equal To...' option is highlighted. A 'Custom AutoFilter' dialog box is also visible, showing the configuration for the 'Amount' column with the condition 'is greater than or equal to 75'.

Product	Price	Quantity	Discount	Amount
Konbu	\$9.00			
Geitost	\$15.00			

Tip

If you disabled the filtering functionality for a particular table, you can reactivate it by using the `TableAutoFilter.Apply` method.

How to: Filter by Cell Values

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Filter by Cell Values](#)

This example demonstrates how to filter data in a column by a list of values.

1. Turn on the filtering functionality for the required range, as described in the [How to: Enable Filtering](#) example.
2. Use the `AutoFilterBase.Columns` property of the `SheetAutoFilter` object to get a collection of columns in the filtered range (the `AutoFilterColumnCollection` object). Each column in the collection is defined by the `AutoFilterColumn` object which provides basic methods for data filtering. To filter data in a particular column, get access to this column by its index in the `AutoFilterColumnCollection` collection.
3. Call the `AutoFilterColumn.ApplyFilterCriteria` method and pass the `FilterValue` object as a parameter. This object defines the value(s) that should be used in the filter criterion. You can use the numeric, boolean, string, [DateTime](#) or `CellValue` types as a filter value. The passed value will be implicitly converted to the **FilterValue** object. In this example, an array of `CellValue` objects is used to display the specific items in the "Product" column.

For an example on how to filter data by a list of date values, refer to the [How to: Filter by Date Values](#) article.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Filter the data in the "Product" column by an array of values.
worksheet.AutoFilter.Columns[1].ApplyFilterCriteria(new CellValue[] { "Mozzarella di Giovanni", "Gorgonzola" });
```

Visual Basic

```
(AutoFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Regional sales")
workbook.Worksheets.ActiveWorksheet = worksheet
' Enable filtering for the specified cell range.
Dim range As Range = worksheet("B2:E23")
worksheet.AutoFilter.Apply(range)
' Filter the data in the "Product" column by an array of values.
worksheet.AutoFilter.Columns(1).ApplyFilterCriteria(New CellValue() { "Mozzarella di Giovanni", "Gorgonzola" })
```

How to: Filter by Date Values

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Filter by Date Values](#)

This example demonstrates how to filter date and time values in a column.

1. Turn on the filtering functionality for the required range, as described in the [How to: Enable Filtering](#) example.
2. Use the `AutoFilterBase.Columns` property of the `SheetAutoFilter` object to get a collection of columns in the filtered range (the `AutoFilterColumnCollection` object). Each column in the collection is defined by the `AutoFilterColumn` object, which provides basic methods for data filtering. To filter data in a particular column, get access to this column by its index in the `AutoFilterColumnCollection` collection.
3. Create a list of date and time values, which should be used in the filter criteria. Each filter value for a date and time filter is defined by an instance of the `DateGrouping` class. Thus, to perform filtering, initialize an instance of the **DateGrouping** class using the `DateGrouping.DateGrouping` constructor with the following parameters.
 - A [DateTime](#) value that specifies the base date or time value to filter by.
 - A `DateTimeGroupingType` enumeration member that specifies the part of the [DateTime](#) value to be used in the filter criteria.

In this example, the **DateGrouping** instance with the `DateGrouping.Value` set to `DateTime(2015, 1, 1)` and `DateGrouping.GroupingType` set to `DateTimeGroupingType.Month` is used to display all reporting dates occurring in January of 2015.

4. To apply a date filter, call the `AutoFilterColumn.ApplyFilterCriteria` method and pass the created list of date grouping items as a parameter.

If a column you wish to filter contains the mixed types of data, you can include additional values to the filter criteria. To do this, use the **ApplyFilterCriteria** method overload with two parameters and pass the `FilterValue` object, containing required cell values, as the first parameter.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Create date grouping item to filter January 2015 dates.
IList<DateGrouping> groupings = new List<DateGrouping>();
DateGrouping dateGroupingJan2015 = new DateGrouping(new DateTime(2015, 1, 1), DateTimeGroupingType.Month);
groupings.Add(dateGroupingJan2015);
// Filter the data in the "Reported Date" column to display values reported in January 2015.
worksheet.AutoFilter.Columns[3].ApplyFilterCriteria("gennaio 2015", groupings);
```

How to: Apply a Custom Date Filter

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Apply a Custom Date Filter](#)

This example demonstrates how to specify the custom filter criteria to display dates that are before, after or equal to the specified date, or between two dates.

1. Turn on the filtering functionality for the required range, as described in the [How to: Enable Filtering](#) example.
2. Use the `AutoFilterBase.Columns` property of the `SheetAutoFilter` object to get a collection of columns in the filtered range (the `AutoFilterColumnCollection` object). Each column in the collection is defined by the `AutoFilterColumn` object, which provides basic methods for data filtering. To filter data in a particular column, get access to this column by its index in the `AutoFilterColumnCollection` collection.
3. To apply a custom complex filter, call the `AutoFilterColumn.ApplyCustomFilter` method and pass the following parameters.
 - A value for the first filter criterion. This value is defined by an instance of the [DateTime](#) structure, which is implicitly converted into the `FilterValue` object. You can also use the `FilterValue.FromDateTime` method to create a filter value from the [DateTime](#) object by using the specified date system.
 - A comparison operator for the first filter criterion. To set an operator, utilize one of the `FilterComparisonOperator` enumeration members. For example, to display dates that are before, after or equal to the specified date, select the `FilterComparisonOperator.LessThanOrEqualTo`, `FilterComparisonOperator.GreaterThanOrEqualTo` or `FilterComparisonOperator.Equal` value, respectively.

If you wish to display date values that are between two specified dates, as shown in the example below, specify the second criterion value by passing some extra parameters to the `AutoFilterColumn.ApplyCustomFilter` method.

- A second `FilterValue` object converted from an instance of the [DateTime](#) structure.
- A comparison operator for the second filter criterion. For the "Between..." operator, the first comparison operator is equal to `FilterComparisonOperator.GreaterThanOrEqualTo`, and the second one is `FilterComparisonOperator.LessThanOrEqualTo`.
- Pass the **true** value as the last parameter of the method to indicate that the **AND** operator should be used to combine the two filter criteria.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Filter values in the "Reported Date" column to display dates that are between June 1, 2014 and February
worksheet.AutoFilter.Columns[3].ApplyCustomFilter(new DateTime(2014, 6, 1), FilterComparisonOperator.Great
```

How to: Apply a Custom Text Filter

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Apply a Custom Text Filter](#)

This example demonstrates how to specify the custom compound filter criteria to filter text values in a column.

1.

Turn on the filtering functionality for the required range, as described in the [How to: Enable Filtering](#) example.
2.

Use the `AutoFilterBase.Columns` property of the `SheetAutoFilter` object to get a collection of columns in the filtered range (the `AutoFilterColumnCollection` object). Each column in the collection is defined by the `AutoFilterColumn` object, which provides basic methods for data filtering. To filter data in a particular column, get access to this column by its index in the `AutoFilterColumnCollection` collection.
3.

To apply a custom complex filter, call the `AutoFilterColumn.ApplyCustomFilter` method overload with five parameters.
 - Specify the first filter criterion. To perform text filtering, use the *wildcard* characters. The asterisk `*` matches any number of characters, while the question mark `?` represents a single character. For example, to display all the products that contain the two-letter combination `"Gi"` in their names, specify the `"*Gi*"` string as the filter criterion value, and use the `FilterComparisonOperator.Equal` member of the `FilterComparisonOperator` enumeration as the comparison operator.

Tip

To filter values containing a specific character, such as the asterisk, question mark or tilde, put the tilde (`~`) before it.

- Specify the second filter criterion. To include blank cells in the filtering results, use the value of the `FilterValue.FilterByBlank` property as the second criterion value, and select the `FilterComparisonOperator.Equal` comparison operator.
- Pass the **false** value as the last parameter of the method to indicate that the **OR** operator should be used to combine the filter criteria specified above.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Filter values in the "Product" column that contain "Gi" and include empty cells.
AutoFilterColumn products = worksheet.AutoFilter.Columns[1];
products.ApplyCustomFilter("*Gi*", FilterComparisonOperator.Equal, FilterValue.FilterByBlank, FilterCompai

Visual Basic

(AutoFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Regional sales")
workbook.Worksheets.ActiveWorksheet = worksheet
' Enable filtering for the specified cell range.
Dim range As Range = worksheet("B2:E23")
worksheet.AutoFilter.Apply(range)
' Filter values in the "Product" column that contain "Gi" and include empt
Dim products As AutoFilterColumn = worksheet.AutoFilter.Columns(1)
products.ApplyCustomFilter("*Gi*", FilterComparisonOperator.Equal, FilterV

How to: Apply a Custom Number Filter

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Apply a Custom Number Filter](#)

This example demonstrates how to specify the custom compound filter criteria to filter numbers in a column.

1.

Turn on the filtering functionality for the required range, as described in the [How to: Enable Filtering](#) example.
2.

Use the `AutoFilterBase.Columns` property of the `SheetAutoFilter` object to get a collection of columns in the filtered range (the `AutoFilterColumnCollection` object). Each column in the collection is defined by the `AutoFilterColumn` object, which provides basic methods for data filtering. To filter data in a particular column, get access to this column by its index in the `AutoFilterColumnCollection` collection.
3.

To apply a custom complex filter, call the `AutoFilterColumn.ApplyCustomFilter` method overload with five parameters.
 - Specify the first filter criterion. Pass **5000** as the filter criterion value, and use the `FilterComparisonOperator.GreaterThanOrEqualTo` member of the `FilterComparisonOperator` enumeration as the comparison operator to find sales values that are greater than or equal to 5000\$.
 - Specify the second filter criterion. Further restrict your data by displaying only values that are less than or equal to 8000\$. To do this, set the second criterion value to **8000**, and select the `FilterComparisonOperator.LessThanOrEqualTo` comparison operator.
 - Pass the **true** value as the last parameter of the method to indicate that the **AND** operator should be used to combine the filter criteria specified above.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Filter values in the "Sales" column that are in a range from 5000$ to 8000$.
AutoFilterColumn sales = worksheet.AutoFilter.Columns[2];
sales.ApplyCustomFilter(5000, FilterComparisonOperator.GreaterThanOrEqualTo, 8000, FilterComparisonOperator
```

Visual Basic

```
(AutoFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Regional sales")
workbook.Worksheets.ActiveWorksheet = worksheet
' Enable filtering for the specified cell range.
Dim range As Range = worksheet("B2:E23")
worksheet.AutoFilter.Apply(range)
' Filter values in the "Sales" column that are in a range from 5000$ to 80
Dim sales As AutoFilterColumn = worksheet.AutoFilter.Columns(2)
sales.ApplyCustomFilter(5000, FilterComparisonOperator.GreaterThanOrEqualTo,
```

How to: Apply a Dynamic Filter

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Apply a Dynamic Filter](#)

This example demonstrates how to apply a dynamic filter to a column.

The dynamic filter allows you to apply one of the predefined date filters to display date values that fall within a specified time period (next, this or last week, month, year, etc.), or to display numbers that are above or below the average.

Note

The dynamic filter criteria can change when the data to which the filter is applied or the current system date changes.

1. Turn on the filtering functionality for the required range, as described in the [How to: Enable Filtering](#) example.
2. Use the AutoFilterBase.Columns property of the SheetAutoFilter object to get a collection of columns in the filtered range (the AutoFilterColumnCollection object). Each column in the collection is defined by the AutoFilterColumn object which provides basic methods for data filtering. To filter data in a particular column, get access to this column by its index in the AutoFilterColumnCollection collection.
3. To apply a dynamic filter, call the AutoFilterColumn.ApplyDynamicFilter method, and pass the appropriate member of the DynamicFilterType enumeration as a parameter depending on the filter type you wish to apply.

In this example, the two filter types are specified. The first filter criterion displays all the sales values in the third column that are above the average. The second filter criterion displays the reporting dates in the fourth column that occurred this year.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Apply a dynamic filter to the "Sales" column to display only values that are above the average.
worksheet.AutoFilter.Columns[2].ApplyDynamicFilter(DynamicFilterType.AboveAverage);
// Apply a dynamic filter to the "Reported Date" column to display values reported this year.
worksheet.AutoFilter.Columns[3].ApplyDynamicFilter(DynamicFilterType.ThisYear);

Visual Basic

(AutoFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Regional sales")
workbook.Worksheets.ActiveWorksheet = worksheet
' Enable filtering for the specified cell range.
Dim range As Range = worksheet("B2:E23")
worksheet.AutoFilter.Apply(range)
' Apply a dynamic filter to the "Sales" column to display only values that
worksheet.AutoFilter.Columns(2).ApplyDynamicFilter(DynamicFilterType.Above
' Apply a dynamic filter to the "Reported Date" column to display values r
worksheet.AutoFilter.Columns(3).ApplyDynamicFilter(DynamicFilterType.ThisY

How to: Filter Top or Bottom Ranked Values

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Filter Top or Bottom Ranked Values](#)

This example demonstrates how to display the top/bottom ranked values using the "Top 10" filter.

1. Turn on the filtering functionality for the required range, as described in the [How to: Enable Filtering](#) example.
2. Use the AutoFilterBase.Columns property of the SheetAutoFilter object to get a collection of columns in the filtered range (the AutoFilterColumnCollection object). Each column in the collection is defined by the AutoFilterColumn object which provides basic methods for data filtering. To filter data in a particular column, get access to this column by its index in the AutoFilterColumnCollection collection.
3. Call the AutoFilterColumn.ApplyTop10Filter and pass the following parameters.
 - A filter operator specified by one of the Top10Type enumeration members.
 - A positive integer that defines the number or percentage value to filter by.

In this example, the AutoFilterColumn.Top10Type property value is equal to Top10Type.Top10Items, and the AutoFilterColumn.Top10Value is set to **10**, so that only the top ten sales values are displayed.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Apply a filter to the "Sales" column to display the top ten values.
worksheet.AutoFilter.Columns[2].ApplyTop10Filter(Top10Type.Top10Items, 10);

How to: Sort Data in the Filtered Range

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Sort Data in the Filtered Range](#)

This example demonstrates how to the sort data in the range to which the filter is applied.

1.

Turn on the filtering functionality for the required range, as described in the [How to: Enable Filtering](#) example.
2.

Use the `AutoFilterBase.SortState` property of the `SheetAutoFilter` object to access the `SortState` object, which contains basic methods to perform sorting in the filtered range. Do one of the following.
- **To sort data by a single column**, call the `SortState.Sort` method, and pass the following parameters: the zero-based index of the column to sort by, and the boolean value that specifies whether you wish to use ascending (**false** value) or descending (**true** value) sort order.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Sort the data in descending order by the first column.
worksheet.AutoFilter.SortState.Sort(0, true);

Visual Basic

(AutoFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Regional sales")
workbook.Worksheets.ActiveWorksheet = worksheet
' Enable filtering for the specified cell range.
Dim range As Range = worksheet("B2:E23")
worksheet.AutoFilter.Apply(range)
' Sort the data in descending order by the first column.
worksheet.AutoFilter.SortState.Sort(0, True)

To sort data by multiple columns, create a list of the `SortCondition` objects, each of which defines the sort criteria to be used. To instantiate a **SortCondition** object, use the `SortCondition.SortCondition` constructor with the following parameters: the index of the column to sort by, and the boolean value that specifies the sort order (**true**, to use the descending order; **false**, to sort in ascending order).

Call the `SortState.Sort` method and pass the list of the created `SortCondition` objects as a parameter. In this example, data is sorted by the first and third columns in descending order.

Note

If you sort data by multiple columns, the next sort order will be applied separately to each group of data that shares the same value in the column used in the previous sort criteria. You can sort by up to 64 columns.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Sort the data in descending order by the first and third columns.
List<SortCondition> sortConditions = new List<SortCondition>();
sortConditions.Add(new SortCondition(0, true));
sortConditions.Add(new SortCondition(2, true));
worksheet.AutoFilter.SortState.Sort(sortConditions);
```

Visual Basic

```
(AutoFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Regional sales")
workbook.Worksheets.ActiveWorksheet = worksheet
' Enable filtering for the specified cell range.
Dim range As Range = worksheet("B2:E23")
worksheet.AutoFilter.Apply(range)
' Sort the data in descending order by the first and third columns.
Dim sortConditions As New List(Of SortCondition)()
sortConditions.Add(New SortCondition(0, True))
sortConditions.Add(New SortCondition(2, True))
worksheet.AutoFilter.SortState.Sort(sortConditions)
```

How to: Reapply a Filter

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Reapply a Filter](#)

The example below demonstrates how to use the `AutoFilterBase.ReApply` method of the `SheetAutoFilter` object to reapply a filter after the data in the filtered range has been changed.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(AutoFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Regional sales"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Enable filtering for the specified cell range.
Range range = worksheet["B2:E23"];
worksheet.AutoFilter.Apply(range);
// Filter values in the "Sales" column that are greater than 5000$.
worksheet.AutoFilter.Columns[2].ApplyCustomFilter(5000, FilterComparisonOperator.GreaterThan);
// Change the data and reapply the filter.
worksheet["D3"].Value = 5000;
worksheet.AutoFilter.ReApply();
```

How to: Clear a Filter

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Filtering](#) > [How to: Clear a Filter](#)

The examples below demonstrate how to clear or disable a filter to redisplay all the rows in the filtered range.

- **Clear a filter from a column**
To remove a filter from a specific column, access this column by its index in the `AutoFilterColumnCollection` collection, and then call the `AutoFilterColumn.Clear` method.

C#

```
// Enable filtering for the specified cell range.  
Range range = worksheet["B2:E23"];  
worksheet.AutoFilter.Apply(range);  
// Filter values in the "Sales" column that are greater than 5000$.  
worksheet.AutoFilter.Columns[2].ApplyCustomFilter(5000, FilterComparisonOperator.GreaterThan);  
// Remove the filter from the SALES column.  
worksheet.AutoFilter.Columns[2].Clear();
```

Visual Basic

```
' Enable filtering for the specified cell range.  
Dim range As Range = worksheet("B2:E23")  
worksheet.AutoFilter.Apply(range)  
' Filter values in the "Sales" column that are greater than 5000$.  
worksheet.AutoFilter.Columns(2).ApplyCustomFilter(5000, FilterComparisonOperator.GreaterThan)  
' Remove the filter from the SALES column.  
worksheet.AutoFilter.Columns(2).Clear()
```

- **Clear all filters**
To clear all the filters specified in a worksheet, use the `AutoFilterBase.Clear` method of the `SheetAutoFilter` object.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(AutoFilterActions.cs)  
Worksheet worksheet = workbook.Worksheets["Regional sales"];  
workbook.Worksheets.ActiveWorksheet = worksheet;  
// Enable filtering for the specified cell range.  
Range range = worksheet["B2:E23"];  
worksheet.AutoFilter.Apply(range);  
// Filter values in the "Sales" column that are greater than 5000$.  
worksheet.AutoFilter.Columns[2].ApplyCustomFilter(5000, FilterComparisonOperator.GreaterThan);  
// Clear the filter.  
worksheet.AutoFilter.Clear();
```

Visual Basic

```
(AutoFilterActions.vb)  
Dim worksheet As Worksheet = workbook.Worksheets("Regional sales")  
workbook.Worksheets.ActiveWorksheet = worksheet  
' Enable filtering for the specified cell range.  
Dim range As Range = worksheet("B2:E23")  
worksheet.AutoFilter.Apply(range)  
' Filter values in the "Sales" column that are greater than 5000$.  
worksheet.AutoFilter.Columns(2).ApplyCustomFilter(5000, FilterComparisonOp  
' Clear the filter.  
worksheet.AutoFilter.Clear()
```

Disable a filter

To disable the filtering functionality, use the `AutoFilterBase.Disable` method of the `SheetAutoFilter` object. In this case, all the specified filters are removed, and the drop-down arrows disappear from the column headers in the filtered range.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#	
<pre>(AutoFilterActions.cs) Worksheet worksheet = workbook.Worksheets["Regional sales"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Enable filtering for the specified cell range. Range range = worksheet["B2:E23"]; worksheet.AutoFilter.Apply(range); // Disable filtering for the entire worksheet. worksheet.AutoFilter.Disable();</pre>	
Visual Basic	
<pre>(AutoFilterActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("Regional sales") workbook.Worksheets.ActiveWorksheet = worksheet ' Enable filtering for the specified cell range. Dim range As Range = worksheet("B2:E23") worksheet.AutoFilter.Apply(range) ' Disable filtering for the entire worksheet. worksheet.AutoFilter.Disable()</pre>	

See Also

[How to: Enable Filtering](#)

Tables

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Tables](#)

This section contains examples that demonstrate how to organize your information in tables, and how to customize the table appearance by using a wide variety of predefined styles or a custom style of your own.

	A	B	C	D	E	F
1						
2		Product ▼	Price ▼	Quantity ▼	Discount ▼	Amount ▼
3		Chocolade	\$5.00	15	3.0%	\$72.75
4		Konbu	\$9.00	55	10.0%	\$445.50
5		Geitost	\$15.00	70	7.0%	\$976.50
6					Total:	\$1,494.75
7						

- [How to: Create a Table](#)
- [How to: Perform Calculations in a Table](#)
- [How to: Apply a Table Style](#)
- [How to: Create, Modify And Delete Table Styles](#)

How to: Create a Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Tables](#) > [How to: Create a Table](#)

The example below demonstrates how to format a range of cells as a table.

1.

To create a table, add a new Table object to the worksheet's collection of tables (Worksheet.Tables) by using the TableCollection.Add method. Pass the following parameters.
 - The range of cells that you wish to format as a table.
 - A Boolean value indicating whether or not the top row of the specified range should be used as the table header.
2.

Format the table by applying one of the built-in table styles. To do this, set the Table.Style property to the table style object from the [Workbook.TableStyles](#) collection. Access the desired style by its **BuiltInTableStyleId** identifier.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

(TableActions.cs)

Worksheet worksheet = workbook.Worksheets[0];
// Insert a table in the worksheet.
Table table = worksheet.Tables.Add(worksheet["A1:F12"], false);
// Format the table by applying a built-in table style.
table.Style = workbook.TableStyles[BuiltInTableStyleId.TableStyleMedium20];

Visual Basic

(TableActions.vb)

Dim worksheet As Worksheet = workbook.Worksheets(0)
' Insert a table in the worksheet.
Dim table As Table = worksheet.Tables.Add(worksheet("A1:F12"), False)
' Format the table by applying a built-in table style.
table.Style = workbook.TableStyles(BuiltInTableStyleId.TableStyleMedium20)

See Also
[How to: Apply a Table Style](#)

How to: Perform Calculations in a Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Tables](#) > [How to: Perform Calculations in a Table](#)

The example below demonstrates how to perform calculations on data in a table using *column names*. The initial table includes a list of products and invoice information on each product: price, quantity and discount. The resulting table will provide an additional column to calculate the amount per product, and an additional row to show the total amount.

	A	B	C	D	E
1					
2		Product ▼	Price ▼	Quantity ▼	Discount ▼
3		Chocolade	5	15	0.03
4		Konbu	9	55	0.1
5		Geitost	15	70	0.07
6					

1. Access table columns by their indexes in the table column collection (TableColumnCollection), returned by the Table.Columns property.
2. Add a new column using the TableColumnCollection.Add method, and set its TableColumn.Name property to "Amount". Headers of other table columns are automatically set to the values of the corresponding cells.
 - table.Columns[0].Name = "Product"
 - table.Columns[1].Name = "Price"
 - table.Columns[2].Name = "Quantity"
 - table.Columns[3].Name = "Discount"
3. Specify the formula to calculate the product amount, and assign it to the "Amount" column using the TableColumn.Formula property. In the formula, refer to table columns by their **names**.
4. Set the Table.ShowTotals property to **true** to display the total row at the bottom of the table.
5. Specify the function to calculate the total amount. To do this, set the TableColumn.TotalRowFunction property of the "Amount" column to TotalRowFunction.Sum.

Tip

In the total row, you can use any formulas you wish, not only functions listed by the TotalRowFunction enumerator. To use a custom formula in the total row, assign it to the TableColumn.TotalRowFormula property of the required table column.

6. Specify number formats to display numbers as currency values in the "Price" and "Amount" columns, and as percentage values in the "Discount" column. To access the data range of a table column, use the TableColumn.DataRange property.

Use the Table.HeaderRowRange and Table.TotalRowRange properties to access table header and total row ranges, and set the alignment.

Change the width of table columns. To do this, access the table range via the Table.Range property, and use its Range.ColumnWidthInCharacters property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(TableActions.cs)
Worksheet worksheet = workbook.Worksheets["TableRanges"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access a table.
Table table = worksheet.Tables[0];
// Access table columns.
TableColumn productColumn = table.Columns[0];
TableColumn priceColumn = table.Columns[1];
TableColumn quantityColumn = table.Columns[2];
TableColumn discountColumn = table.Columns[3];
// Add a new column to the end of the table .
TableColumn amountColumn = table.Columns.Add();
// Set the name of the last column.
amountColumn.Name = "Amount";
// Set the formula to calculate the amount per product
// and display results in the "Amount" column.
amountColumn.Formula = "=[Price]*[Quantity]*(1-[Discount])";
// Display the total row in the table.
table.ShowTotals = true;
// Set the label and function to display the sum of the "Amount" column.
discountColumn.TotalRowLabel = "Total:";
amountColumn.TotalRowFunction = TotalRowFunction.Sum;
// Specify the number format for each column.
priceColumn.DataRange.NumberFormat = "$#,##0.00";
discountColumn.DataRange.NumberFormat = "0.0%";
amountColumn.Range.NumberFormat = "$#,##0.00;$#,##0.00;\"\\\";@";
// Specify horizontal alignment for header and total rows of the table.
table.HeaderRowRange.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center;
table.TotalRowRange.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center;
// Specify horizontal alignment to display data in all columns except the first one.
for (int i = 1; i < table.Columns.Count; i++)
{
    table.Columns[i].DataRange.Alignment.Horizontal = SpreadsheetHorizontalAlignment.Center;
}
// Set the width of table columns.
table.Range.ColumnWidthInCharacters = 10;
worksheet.Visible = true;
```

Visual Basic

```

(TableActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("TableRanges")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access a table.
Dim table As Table = worksheet.Tables(0)
' Access table columns.
Dim productColumn As TableColumn = table.Columns(0)
Dim priceColumn As TableColumn = table.Columns(1)
Dim quantityColumn As TableColumn = table.Columns(2)
Dim discountColumn As TableColumn = table.Columns(3)
' Add a new column to the end of the table .
Dim amountColumn As TableColumn = table.Columns.Add()
' Set the name of the last column.
amountColumn.Name = "Amount"
' Set the formula to calculate the amount per product
' and display results in the "Amount" column.
amountColumn.Formula = "=[Price]*[Quantity]*(1-[Discount])"
' Display the total row in the table.
table.ShowTotals = True
' Set the label and function to display the sum of the "Amount" column.
discountColumn.TotalRowLabel = "Total:"
amountColumn.TotalRowFunction = TotalRowFunction.Sum
' Specify the number format for each column.
priceColumn.DataRange.NumberFormat = "$#,##0.00"
discountColumn.DataRange.NumberFormat = "0.0%"
amountColumn.Range.NumberFormat = "$#,##0.00;$#,##0.00;\"";@\"
' Specify horizontal alignment for header and total rows of the table.
table.HeaderRowRange.Alignment.Horizontal = SpreadsheetHorizontalAlignment
table.TotalRowRange.Alignment.Horizontal = SpreadsheetHorizontalAlignment.
' Specify horizontal alignment to display data in all columns except the f
For i As Integer = 1 To table.Columns.Count - 1
    table.Columns(i).DataRange.Alignment.Horizontal = SpreadsheetHorizonta
Next i
' Set the width of table columns.
table.Range.ColumnWidthInCharacters = 10
worksheet.Visible = True

```

The image below illustrates the result (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F
1						
2		Product ▾	Price ▾	Quantity ▾	Discount ▾	Amount ▾
3		Chocolade	\$5.00	15	3.0%	\$72.75
4		Konbu	\$9.00	55	10.0%	\$445.50
5		Geitost	\$15.00	70	7.0%	\$976.50
6					Total:	\$1,494.75
7						

See Also

[How to: Create a Table](#)

How to: Apply a Table Style

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Tables](#) > [How to: Apply a Table Style](#)

The example below demonstrates how to format a table by using one of the predefined table styles.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

To format a table by applying a table style, assign the corresponding TableStyle object to the Table.Style property. Access the required table style object from the [Workbook.TableStyles](#) collection by the table style name. Built-in table styles can also be obtained by their ids (the **BuiltInTableStyleId** enumeration members).

A table style consists of the collection of table style elements (TableStyle.TableStyleElements). Each table style element (TableStyleElement) specifies the formatting for a particular element of a table. The TableStyleElementType enumerator lists supported table style element types. The Table object provides the following properties to optionally specify table elements to be formatted as defined by the corresponding elements of the applied table style.

- Table.ShowHeaders - shows the header row in the table and formats it as specified by the TableStyleElementType.HeaderRow element of the applied table style.
- Table.ShowTotals - shows the total row in the table and formats it as specified by the TableStyleElementType.TotalRow element of the applied table style.
- Table.ShowTableStyleRowStripes - applies striped row formatting to a table as specified by the TableStyleElementType.FirstRowStripe and TableStyleElementType.SecondRowStripe elements of the applied table style.
- Table.ShowTableStyleColumnStripes - applies striped column formatting to a table as specified by the TableStyleElementType.FirstColumnStripe and TableStyleElementType.SecondColumnStripe elements of the applied table style.
- Table.ShowTableStyleFirstColumn - formats the first column of the table as specified by the TableStyleElementType.FirstColumn, TableStyleElementType.FirstHeaderCell and TableStyleElementType.FirstTotalCell elements of the applied table style.
- Table.ShowTableStyleLastColumn - formats the last column of the table as specified by the TableStyleElementType.LastColumn, TableStyleElementType.LastHeaderCell and TableStyleElementType.LastTotalCell elements of the applied table style.

This example demonstrates how to access a table style by its name and apply it to an existing table. The Table.ShowHeaders and Table.ShowTotals properties are set to **true** to show the header and total row of the table. The Table.ShowTableStyleRowStripes and Table.ShowTableStyleColumnStripes properties are used to apply the striped column formatting to the table.

The Table.ShowTableStyleFirstColumn property is also set to **true** to apply specific formatting to the first column of the table.

C#

```
(TableActions.cs)
Worksheet worksheet = workbook.Worksheets["FormatTable"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access a table.
Table table = worksheet.Tables[0];
// Access the workbook's collection of table styles.
TableStyleCollection tableStyles = workbook.TableStyles;
// Access the built-in table style from the collection by its name.
TableStyle tableStyle = tableStyles[BuiltInTableStyleId.TableStyleMedium16];
// Apply the table style to the existing table.
table.Style = tableStyle;
// Show header and total rows and format them as specified by the applied table style.
table.ShowHeaders = true;
table.ShowTotals = true;
// Apply banded column formatting to the table.
table.ShowTableStyleRowStripes = false;
table.ShowTableStyleColumnStripes = true;
// Apply special formatting to the first column of the table.
table.ShowTableStyleFirstColumn = true;
worksheet.Visible = true;
```

Visual Basic

```
(TableActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("FormatTable")
worksheet.Worksheets.ActiveWorksheet = worksheet
' Access a table.
Dim table As Table = worksheet.Tables(0)
' Access the workbook's collection of table styles.
Dim tableStyles As TableStyleCollection = workbook.TableStyles
' Access the built-in table style from the collection by its name.
Dim tableStyle As TableStyle = tableStyles(BuiltInTableStyleId.TableStyleM
' Apply the table style to the existing table.
table.Style = tableStyle
' Show header and total rows and format them as specified by the applied t
table.ShowHeaders = True
table.ShowTotals = True
' Apply banded column formatting to the table.
table.ShowTableStyleRowStripes = False
table.ShowTableStyleColumnStripes = True
' Apply special formatting to the first column of the table.
table.ShowTableStyleFirstColumn = True
worksheet.Visible = True
```

The image below shows the table appearance when the *TableStyleMedium16* style is applied (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F	
1							
2		Product ▾	Price ▾	Quantity ▾	Discount ▾	Amount ▾	
3		Chocolade	\$5.00	15	3.0%	\$72.75	
4		Konbu	\$9.00	55	10.0%	\$445.50	
5		Geitost	\$15.00	70	7.0%	\$976.50	
6					Total:	\$1,494.75	
7							

See Also
[How to: Create a Table](#)
[How to: Create, Modify And Delete Table Styles](#)

How to: Create, Modify And Delete Table Styles

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Tables](#) > [How to: Create, Modify And Delete Table Styles](#)

A workbook contains a collection of table styles to format tables (see the [How to: Apply a Table Style](#) example). The [Workbook.TableStyles](#) property returns the TableStyleCollection object, which specifies the workbook's collection of table styles. By default, this collection contains built-in table styles similar to Microsoft® Excel® and the *None* table style, which specifies that no formatting should be applied to the table. If you create custom table styles, they are also placed in the workbook's table style collection.

You can do the following to manage the workbook's collection of table styles.


- [Modify existing table styles](#) (excluding built-in table styles)
- [Add new custom table styles](#)
- [Duplicate existing table styles](#)
- [Delete existing table styles](#) (excluding built-in table styles)

Modify Existing Table Styles

A table style (TableStyle) consists of a collection of table style elements (TableStyle.TableStyleElements). Each table style element (TableStyleElement) specifies formatting for a particular element of a table. The TableStyleElementType enumerator lists the supported table style element types.

Use properties of the TableStyleElement object to customize borders (TableStyleElement.Borders), fill (TableStyleElement.Fill) and font (TableStyleElement.Font) for the corresponding table element. If you wish to create a style providing striped row or column formatting for a table, customize the *FirstRowStripe*, *SecondRowStripe*, *FirstColumnStripe* or *SecondColumnStripe* table style elements and set their TableStyleElement.StripeSize property, which specifies the banding rule.

Thus, to modify a table style, follow the steps below.

1. Access the table style to be changed. To do this, get the corresponding TableStyle object from the [Workbook.TableStyles](#) collection by the table style name.
-  **Important**

Built-in table styles cannot be modified. To check whether a table style is built-in or custom, use the TableStyle.BuiltIn property.
2. Call the TableStyle.BeginUpdate method.
 3. Access the table style element to be modified from the TableStyle.TableStyleElements collection by the corresponding TableStyleElementType enumeration member. Use the TableStyleElement properties to specify the required formatting for the element. If you need to remove existing formatting from the element, use its TableStyleElement.Clear method.

Repeat this step for all table style elements you wish to modify.

4. Call the TableStyle.EndUpdate method.

C#

```
using DevExpress.Spreadsheet;
// ...
// Access the table style to be modified.
TableStyle tableStyle = workbook.TableStyles["tableStyleName"];
// Change the required formatting characteristics of the style elements.
tableStyle.BeginUpdate();
try {
    TableStyleElement wholeTable = tableStyle.TableStyleElements[TableStyleElementType.WholeTable];
    // wholeTable.Fill...
    // wholeTable.Borders...
    // wholeTable.Font...
    TableStyleElement tableHeader = tableStyle.TableStyleElements[TableStyleElementType.HeaderRow];
    // tableHeader.Fill.BackgroundColor...
    // tableHeader.Font...
    TableStyleElement firstColumn = tableStyle.TableStyleElements[TableStyleElementType.FirstColumn];
    // firstColumn.Clear();
    // ...
}
finally {
    tableStyle.EndUpdate();
}
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
' Access the table style to be modified.
Dim tableStyle As TableStyle = workbook.TableStyles("testTableStyle")
' Change the required formatting characteristics of the style elements.
tableStyle.BeginUpdate()
Try
    Dim wholeTable As TableStyleElement = tableStyle.TableStyleElements(TableStyleElementType.WholeTable)
    ' wholeTable.Fill...
    ' wholeTable.Borders...
    ' wholeTable.Font...
    Dim tableHeader As TableStyleElement = tableStyle.TableStyleElements(TableStyleElementType.HeaderRow)
    ' tableHeader.Fill.BackgroundColor...
    ' tableHeader.Font...
    Dim firstColumn As TableStyleElement = tableStyle.TableStyleElements(TableStyleElementType.FirstColumn)
    ' firstColumn.Clear();
    ' ...
Finally
    tableStyle.EndUpdate()
End Try
```

After a table style is changed, the style modifications are automatically applied to all tables that use this style. For details on how a table style is applied to a table and table elements, review the [How to: Apply a Table Style](#) document.

Create Your Own Custom Table Style

• Create a New Table Style

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

The example below demonstrates how to create a custom style to format tables.

1.
- Add a new table style to the [Workbook.TableStyles](#) collection by calling the `TableStyleCollection.Add` method with the table style name passed as a parameter. This method returns the `TableStyle` object that represents the newly created table style. This object's `TableStyle.Name` property is set to the specified name. Other settings are identical to the *None* table style (the default built-in table style that specifies no formatting for a table).

Important

Note that table styles have unique names in the collection. To ensure that there is no table style under the specified name in the collection, use the `TableStyleCollection.Contains` method.

2.
- Modify elements of the created table style (`TableStyle.TableStyleElements`) within the `TableStyle.BeginUpdate` and `TableStyle.EndUpdate` paired methods.

C#

```
(TableActions.cs)
Worksheet worksheet = workbook.Worksheets["Custom Table Style"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access a table.
Table table = worksheet.Tables[0];
String styleName = "testTableStyle";
// If the style under the specified name already exists in the collection,
if (workbook.TableStyles.Contains(styleName))
{
    // apply this style to the table.
    table.Style = workbook.TableStyles[styleName];
}
else
{
    // Add a new table style under the "testTableStyle" name to the TableStyles collection.
    TableStyle customTableStyle = workbook.TableStyles.Add("testTableStyle");
    // Modify the required formatting characteristics of the table style.
    // Specify the format for different table elements.
    customTableStyle.BeginUpdate();
    try
    {
        customTableStyle.TableStyleElements[TableStyleElementType.WholeTable].Font.Color = Color.FromArgb
        // Specify formatting characteristics for the table header row.
        TableStyleElement headerRowStyle = customTableStyle.TableStyleElements[TableStyleElementType.HeaderRow];
        headerRowStyle.Fill.BackgroundColor = Color.FromArgb(64, 66, 166);
        headerRowStyle.Font.Color = Color.White;
        headerRowStyle.Font.Bold = true;
        // Specify formatting characteristics for the table total row.
        TableStyleElement totalRowStyle = customTableStyle.TableStyleElements[TableStyleElementType.TotalRow];
        totalRowStyle.Fill.BackgroundColor = Color.FromArgb(115, 193, 211);
        totalRowStyle.Font.Color = Color.White;
        totalRowStyle.Font.Bold = true;
        // Specify banded row formatting for the table.
        TableStyleElement secondRowStripeStyle = customTableStyle.TableStyleElements[TableStyleElementType.SecondRowStripe];
        secondRowStripeStyle.Fill.BackgroundColor = Color.FromArgb(234, 234, 234);
        secondRowStripeStyle.StripeSize = 1;
    }
    finally
    {
        customTableStyle.EndUpdate();
    }
    // Apply the created custom style to the table.
    table.Style = customTableStyle;
}
worksheet.Visible = true;
```

Visual Basic

```
(TableActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Custom Table Style")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access a table.
Dim table As Table = worksheet.Tables(0)
Dim styleName As String = "testTableStyle"
' If the style under the specified name already exists in the collection,
If workbook.TableStyles.Contains(styleName) Then
    ' apply this style to the table.
    table.Style = workbook.TableStyles(styleName)
Else
    ' Add a new table style under the "testTableStyle" name to the TableStyles collection.
    Dim customTableStyle_Renamed As TableStyle = workbook.TableStyles.Add("testTableStyle")
    ' Modify the required formatting characteristics of the table style.
    ' Specify the format for different table elements.
    customTableStyle_Renamed.BeginUpdate()
    Try
        customTableStyle_Renamed.TableStyleElements(TableStyleElementType.WholeTable).Font.Color = Color.Red
        ' Specify formatting characteristics for the table header row.
        Dim headerRowStyle As TableStyleElement = customTableStyle_Renamed.TableStyleElements(TableStyleElementHeader)
        headerRowStyle.Fill.BackgroundColor = Color.FromArgb(64, 66, 166)
        headerRowStyle.Font.Color = Color.White
        headerRowStyle.Font.Bold = True
        ' Specify formatting characteristics for the table total row.
        Dim totalRowStyle As TableStyleElement = customTableStyle_Renamed.TableStyleElements(TableStyleElementTotal)
        totalRowStyle.Fill.BackgroundColor = Color.FromArgb(115, 193, 211)
        totalRowStyle.Font.Color = Color.White
        totalRowStyle.Font.Bold = True
        ' Specify banded row formatting for the table.
        Dim secondRowStripeStyle As TableStyleElement = customTableStyle_Renamed.TableStyleElements(TableStyleElementSecondRowStripe)
        secondRowStripeStyle.Fill.BackgroundColor = Color.FromArgb(234, 234, 234)
        secondRowStripeStyle.StripeSize = 1
    Finally
        customTableStyle_Renamed.EndUpdate()
    End Try
    ' Apply the created custom style to the table.
    table.Style = customTableStyle_Renamed
End If
worksheet.Visible = True
```

The following image shows the table formatted with the custom table style created by the code above (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F
1						
2		Product	Price	Quantity	Discount	Amount
3		Chocolade	\$5.00	15	3.0%	\$72.75
4		Konbu	\$9.00	55	10.0%	\$445.50
5		Geitost	\$15.00	70	7.0%	\$976.50
6		Total:				\$1,494.75
7						

Duplicate an Existing Table Style

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

You can create new custom table styles based on existing table styles (for example, based on built-in table styles). To do this, use the `TableStyle.Duplicate` method. This method creates a copy of the specified style and returns the `TableStyle` object representing the newly created style. You can use properties of this object to change the style format settings.

For example, the code snippet below demonstrates how to duplicate an existing style and modify the new style a bit by changing the color of the header row at the top of the table.

C#
<pre>(TableActions.cs) Worksheet worksheet = workbook.Worksheets["Duplicate Table Style"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Access a table. Table table1 = worksheet.Tables[0]; Table table2 = worksheet.Tables[1]; // Get the table style to be duplicated. TableStyle sourceTableStyle = workbook.TableStyles[BuiltInTableStyleId.Tab // Duplicate the table style. TableStyle newTableStyle = sourceTableStyle.Duplicate(); // Modify the required formatting characteristics of the created table sty // For example, change background color of the table header. newTableStyle.TableStyleElements[TableStyleElementType.HeaderRow].Fill.Bac table1.Style = sourceTableStyle; table2.Style = newTableStyle; worksheet.Visible = true;</pre>


Visual Basic
<pre>(TableActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("Duplicate Table Style") workbook.Worksheets.ActiveWorksheet = worksheet ' Access a table. Dim table1 As Table = worksheet.Tables(0) Dim table2 As Table = worksheet.Tables(1) ' Get the table style to be duplicated. Dim sourceTableStyle As TableStyle = workbook.TableStyles(BuiltInTableStyl ' Duplicate the table style. Dim newTableStyle As TableStyle = sourceTableStyle.Duplicate() ' Modify the required formatting characteristics of the created table styl ' For example, change background color of the table header. newTableStyle.TableStyleElements(TableStyleElementType.HeaderRow).Fill.Bac table1.Style = sourceTableStyle table2.Style = newTableStyle worksheet.Visible = True</pre>

The image below shows the *TableStyleMedium17* built-in table style and its modified copy (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F	
1		Built-In Table Style					
2							
3		Product ▾	Price ▾	Quantity ▾	Discount ▾	Amount ▾	
4		Chocolade	\$5.00	15	3.0%	\$72.75	
5		Konbu	\$9.00	55	10.0%	\$445.50	
6		Geitost	\$15.00	70	7.0%	\$976.50	
7					Total:	\$1,494.75	
8							
9							
10		Custom Table Style Based on the Built-In Table Style					
11							
12		Product ▾	Price ▾	Quantity ▾	Discount ▾	Amount ▾	
13		Chocolade	\$5.00	15	3.0%	\$72.75	
14		Konbu	\$9.00	55	10.0%	\$445.50	
15		Geitost	\$15.00	70	7.0%	\$976.50	
16					Total:	\$1,494.75	
17							

Delete Table Styles

To remove a table style from the [Workbook.TableStyles](#) collection, use the `TableStyleCollection.Remove` method. After a table style is deleted, all tables to which this style is applied will be formatted with the default style (`TableStyleCollection.DefaultStyle`).

 **Important**

Built-in table styles cannot be removed. To check whether a table style is built-in or custom, use the `TableStyle.BuiltIn` property.

See Also

- [How to: Create a Table](#)
- [How to: Apply a Table Style](#)

Pivot Tables

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#)

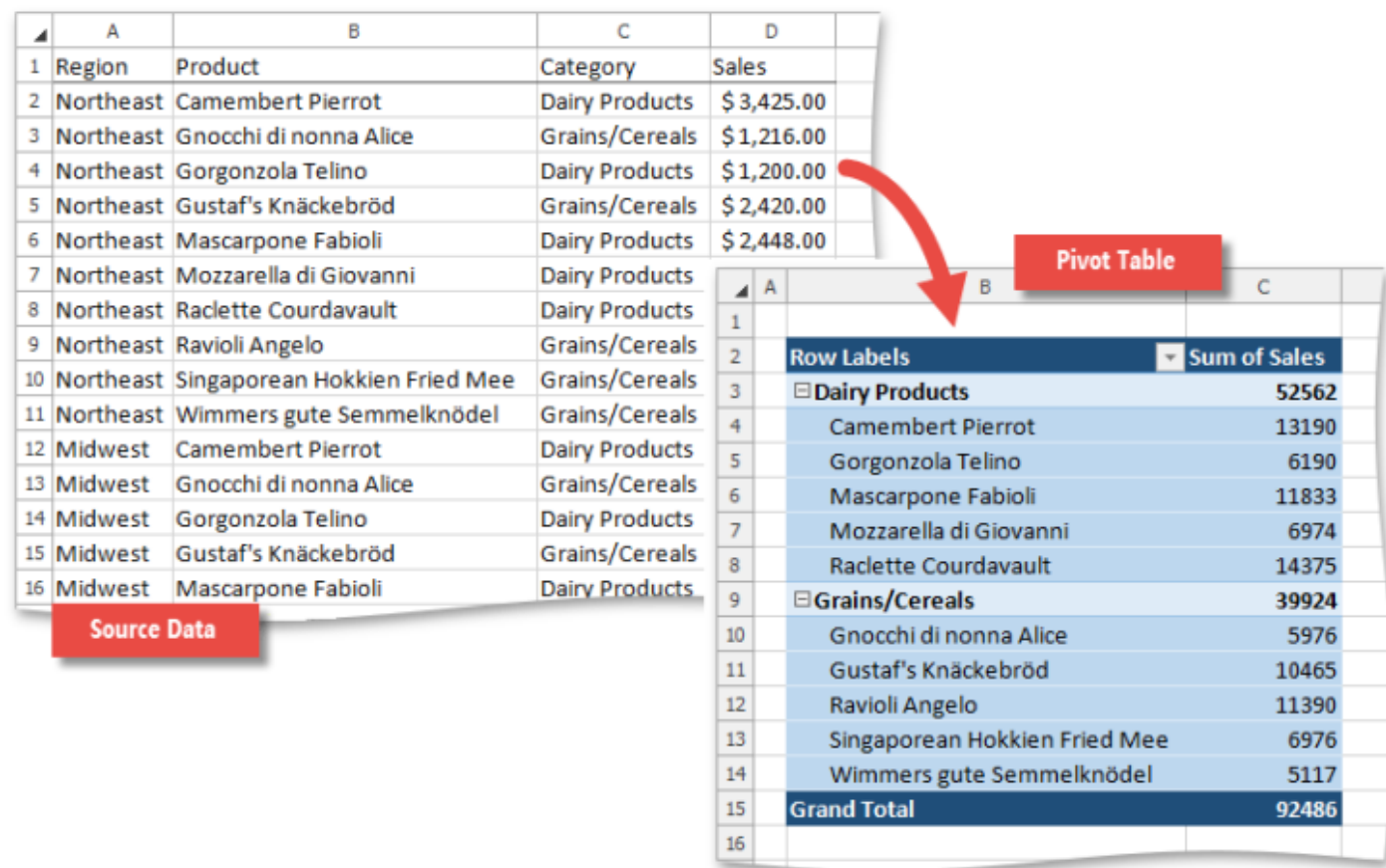
This section contains the following examples:

- [How to: Create a Pivot Table](#)
- [How to: Refresh a Pivot Table](#)
- [How to: Change a Data Source for a Pivot Table](#)
- [How to: Move a Pivot Table](#)
- [How to: Clear or Remove a Pivot Table](#)
- [How to: Change the PivotTable Layout](#)
- [How to: Subtotal Fields in a Pivot Table](#)
- [How to: Display or Hide Grand Totals for a Pivot Table](#)
- [How to: Apply a Predefined Style to a Pivot Table](#)
- [How to: Apply a Custom Style to a Pivot Table](#)
- [How to: Control Style Options](#)
- [How to: Change the Summary Function for a Data Field](#)
- [How to: Apply a Custom Calculation to a Data Field](#)
- [How to: Create a Calculated Field](#)
- [How to: Create a Calculated Item](#)
- [How to: Sort Items in a Pivot Table](#)
- [How to: Filter Items in a Pivot Table](#)
- [How to: Group Items in a Pivot Table](#)

How to: Create a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Create a Pivot Table](#)


The following examples demonstrate how to use the [Pivot Tables](#) to create a pivot table, which summarizes data in a [cell range](#).



To create a new pivot table, use the PivotTableCollection.Add method of the Worksheet.PivotTables collection accessed for a worksheet where the report should be located. You can [use a cell range as the data source](#) for your PivotTable report, or [base it on the data cache of the existing pivot table](#) (for details, see PivotCache).

To fill the created pivot table with data, add the necessary [fields](#) to it. All pivot fields are stored in the PivotFieldCollection collection returned by the PivotTable.Fields property. To add a field to the PivotTable report, access the required field by its name in the collection (by default, field names originate from the column labels in the source range) and move it to one of four PivotTable areas, listed in the table below.

To add a field to	Do this
Row axis area 	Use the PivotFieldReferenceCollection.Add method of the PivotTable.RowFields collection.
Column axis area 	Use the PivotFieldReferenceCollection.Add method of the PivotTable.ColumnFields collection.
Report filter area 	Use the PivotPageFieldCollection.Add method of the PivotTable.PageFields collection.

Data area 	Use the PivotDataFieldCollection.Add method of the PivotTable.DataFields collection.
---	--

Create a Pivot Table Using a Cell Range as the Data Source

Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T501894 .	

C#	
<pre>(PivotTableActions.cs) Worksheet sourceWorksheet = workbook.Worksheets["Data1"]; Worksheet worksheet = workbook.Worksheets.Add(); workbook.Worksheets.ActiveWorksheet = worksheet; // Create a pivot table using the cell range "A1:D41" as the data source. PivotTable pivotTable = worksheet.PivotTables.Add(sourceWorksheet["A1:D41"], worksheet["B2"]); // Add the "Category" field to the row axis area. pivotTable.RowFields.Add(pivotTable.Fields["Category"]); // Add the "Product" field to the row axis area. pivotTable.RowFields.Add(pivotTable.Fields["Product"]); // Add the "Sales" field to the data area. pivotTable.DataFields.Add(pivotTable.Fields["Sales"]);</pre>	

Visual Basic	
<pre>(PivotTableActions.vb) Dim sourceWorksheet As Worksheet = workbook.Worksheets("Data1") Dim worksheet As Worksheet = workbook.Worksheets.Add() workbook.Worksheets.ActiveWorksheet = worksheet ' Create a pivot table using the cell range "A1:D41" as the data source. Dim pivotTable As PivotTable = worksheet.PivotTables.Add(sourceWorksheet("A1:D41"), worksheet("B2")) ' Add the "Category" field to the row axis area. pivotTable.RowFields.Add(pivotTable.Fields("Category")) ' Add the "Product" field to the row axis area. pivotTable.RowFields.Add(pivotTable.Fields("Product")) ' Add the "Sales" field to the data area. pivotTable.DataFields.Add(pivotTable.Fields("Sales"))</pre>	

Create a Pivot Table based on the PivotTable Cache

Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T501894 .	
C#	

```
(PivotTableActions.cs)
Worksheet worksheet = workbook.Worksheets.Add();
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a pivot table based on the specified PivotTable cache.
PivotCache cache = workbook.Worksheets["Report1"].PivotTables["PivotTable1"];
PivotTable pivotTable = worksheet.PivotTables.Add(cache, worksheet["B2"]);
// Add the "Category" field to the row axis area.
pivotTable.RowFields.Add(pivotTable.Fields["Category"]);
// Add the "Product" field to the row axis area.
pivotTable.RowFields.Add(pivotTable.Fields["Product"]);
// Add the "Sales" field to the data area.
pivotTable.DataFields.Add(pivotTable.Fields["Sales"]);
// Set the default style for the pivot table.
pivotTable.Style = workbook.TableStyles.DefaultPivotStyle;
```

Visual Basic
<pre>(PivotTableActions.vb) Dim worksheet As Worksheet = workbook.Worksheets.Add() workbook.Worksheets.ActiveWorksheet = worksheet ' Create a pivot table based on the specified PivotTable cache. Dim cache As PivotCache = workbook.Worksheets("Report1").PivotTables("PivotTable1") Dim pivotTable As PivotTable = worksheet.PivotTables.Add(cache, worksheet["B2"]) ' Add the "Category" field to the row axis area. pivotTable.RowFields.Add(pivotTable.Fields("Category")) ' Add the "Product" field to the row axis area. pivotTable.RowFields.Add(pivotTable.Fields("Product")) ' Add the "Sales" field to the data area. pivotTable.DataFields.Add(pivotTable.Fields("Sales")) ' Set the default style for the pivot table. pivotTable.Style = workbook.TableStyles.DefaultPivotStyle</pre>

PivotTable Report Limitations

While creating pivot tables, take into account the following restrictions.

Feature	Limit
Number of pivot tables on a worksheet	Limited by available memory
Number of row fields	Limited by available memory
Number of column fields	Limited by available memory
Number of page fields	256
Number of data fields	256
Number of unique items per field	1,048,576 for XLSX format 32,500 for XLS format

How to: Refresh a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Refresh a Pivot Table](#)

To refresh a pivot table and retrieve the latest data from a data source, use the PivotCache.Refresh method of the PivotTable cache accessible through the PivotTable.Cache property. All pivot tables that are based on the same PivotCache will also be refreshed.

Note

If your PivotTable report contains [cell formatting](#), it may disappear after the refresh operation.

If you wish to prevent column widths from being automatically adjusted when the PivotTable data is refreshed, set the PivotBehaviorOptions.AutoFitColumns property to **false**.

C#

```
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Refresh the pivot table.
pivotTable.Cache.Refresh();
```

Visual Basic

```
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Refresh the pivot table.
pivotTable.Cache.Refresh()
```

To refresh all pivot tables in a workbook at once, use the PivotCacheCollection.RefreshAll method.

How to: Change a Data Source for a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Change a Data Source for a Pivot Table](#)

To change a data source for a PivotTable report, use the PivotTable.ChangeDataSource method overloads. You can update your pivot table to reflect changes in the initial source range (e.g., if it was extended by new rows or columns, or existing records or fields were removed), or rebuild the report based on a completely different cell range, as shown in the example below. In the latter case, you should re-create the pivot table and fill it with new data by adding necessary [fields](#) to it.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
Worksheet sourceWorksheet = workbook.Worksheets["Data2"];
// Change the data source of the pivot table.
pivotTable.ChangeDataSource(sourceWorksheet["A1:H6367"]);
// Add the "State" field to the row axis area.
pivotTable.RowFields.Add(pivotTable.Fields["State"]);
// Add the "Yearly Earnings" field to the data area.
PivotDataField dataField = pivotTable.DataFields.Add(pivotTable.Fields["Yearly Earnings"]);
// Calculate the average of the "Yearly Earnings" values for each state.
dataField.SummarizeValuesBy = PivotDataConsolidationFunction.Average;
```

Visual Basic

```
(PivotTableActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
Dim sourceWorksheet As Worksheet = workbook.Worksheets("Data2")
' Change the data source of the pivot table.
pivotTable.ChangeDataSource(sourceWorksheet("A1:H6367"))
' Add the "State" field to the row axis area.
pivotTable.RowFields.Add(pivotTable.Fields("State"))
' Add the "Yearly Earnings" field to the data area.
Dim dataField As PivotDataField = pivotTable.DataFields.Add(pivotTable.Fie
' Calculate the average of the "Yearly Earnings" values for each state.
dataField.SummarizeValuesBy = PivotDataConsolidationFunction.Average
```

How to: Move a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Move a Pivot Table](#)

To move a PivotTable report, use the PivotTable.MoveTo method. You can place a pivot table in another location in the existing worksheet or move it to a new worksheet.

Note

If a cell range where you wish to place your report is a regular range containing data, it will be overwritten without warning. But if the destination cell range contains a pivot table or [table](#), the **System.InvalidOperationException** will be thrown since the pivot table cannot overlap another table or a PivotTable report.

Move a Pivot Table to Another Location in the Existing Worksheet

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

(PivotTableActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Change the pivot table location.
worksheet.PivotTables["PivotTable1"].MoveTo(worksheet["A7"]);
// Refresh the pivot table.
worksheet.PivotTables["PivotTable1"].Cache.Refresh();

Visual Basic

(PivotTableActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Change the pivot table location.
worksheet.PivotTables("PivotTable1").MoveTo(worksheet("A7"))
' Refresh the pivot table.
worksheet.PivotTables("PivotTable1").Cache.Refresh()

Move a Pivot Table to a New Worksheet

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

(PivotTableActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
// Create a new worksheet.
Worksheet targetWorksheet = workbook.Worksheets.Add();
// Access the pivot table by its name in the collection
// and move it to the new worksheet.
worksheet.PivotTables["PivotTable1"].MoveTo(targetWorksheet["B2"]);
// Refresh the pivot table.
targetWorksheet.PivotTables["PivotTable1"].Cache.Refresh();
workbook.Worksheets.ActiveWorksheet = targetWorksheet;

Visual Basic

```
(PivotTableActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
' Create a new worksheet.
Dim targetWorksheet As Worksheet = workbook.Worksheets.Add()
' Access the pivot table by its name in the collection
' and move it to the new worksheet.
worksheet.PivotTables("PivotTable1").MoveTo(targetWorksheet("B2"))
' Refresh the pivot table.
targetWorksheet.PivotTables("PivotTable1").Cache.Refresh()
workbook.Worksheets.ActiveWorksheet = targetWorksheet
```

How to: Clear or Remove a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Clear or Remove a Pivot Table](#)

The examples below demonstrate how to clear a pivot table by removing all its fields or completely delete a report from a worksheet.

Select the task you wish to perform.

- [Clear a Pivot Table](#)
- [Delete a Pivot Table](#)

Clear a Pivot Table

To remove all fields and formatting from a PivotTable report, use the PivotTable.Clear method. This method resets the pivot table to the initial state before any fields are added to it, but does not delete the report. The data connection, PivotCache, and the report location are preserved.

Note

If a pivot table you wish to clear is based on the same data as other pivot tables in a document, the PivotTable.Clear method will also remove [grouping](#), [calculated fields](#), and [calculated items](#) from other shared PivotTable reports.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

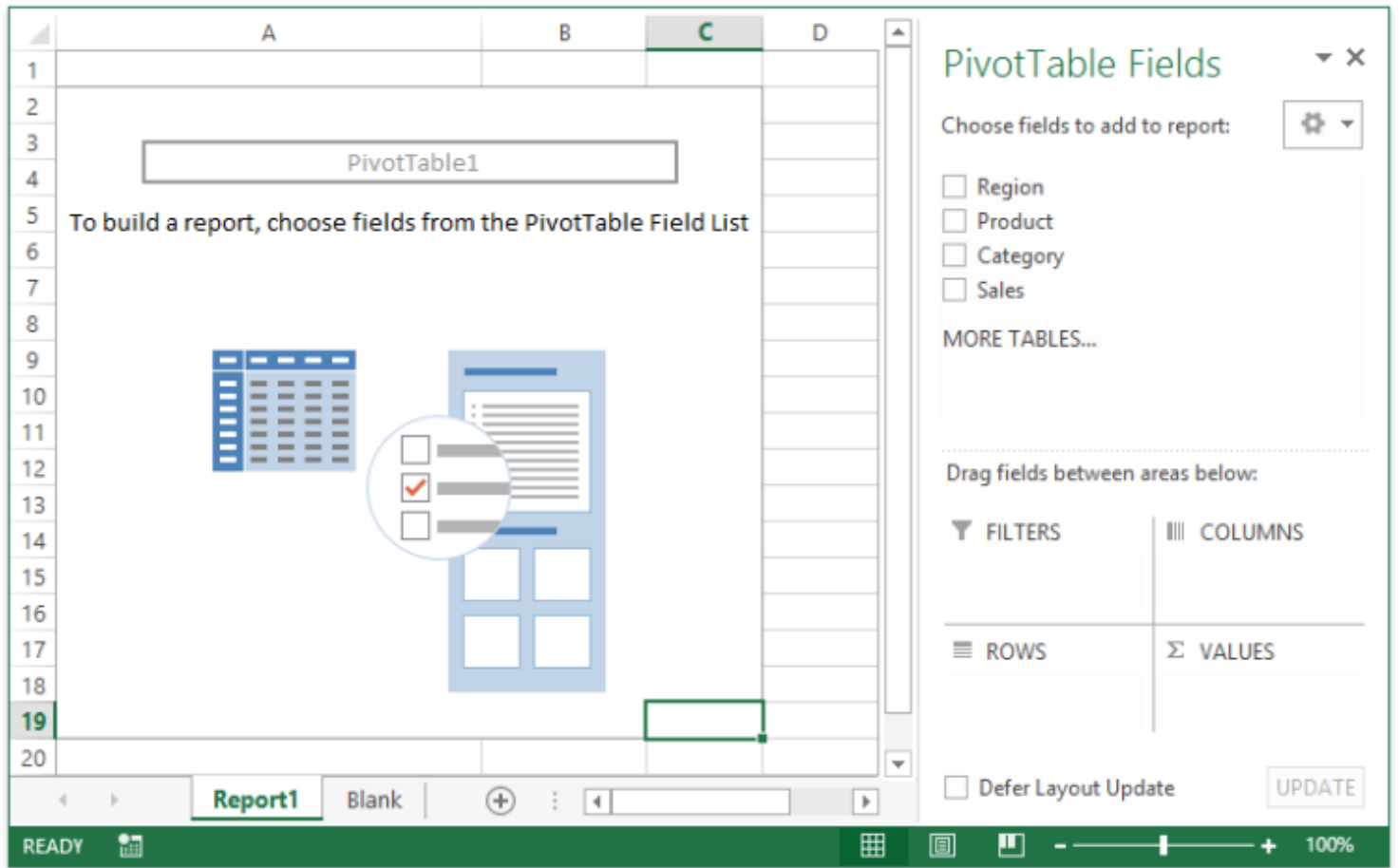
C#

(PivotTableActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Clear the pivot table.
worksheet.PivotTables["PivotTable1"].Clear();

Visual Basic

(PivotTableActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Clear the pivot table.
worksheet.PivotTables("PivotTable1").Clear()

The following image shows the result of the code's execution (the workbook is opened in Microsoft® Excel®). A blank pivot table is displayed so you can start designing the report's layout all over again.



Delete a Pivot Table

To delete a PivotTable report from a worksheet, remove it from the worksheet's PivotTableCollection by using the PivotTableCollection.Remove or PivotTableCollection.RemoveAt method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#	
<pre>(PivotTableActions.cs) Worksheet worksheet = workbook.Worksheets["Report1"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Access the pivot table by its name in the collection. PivotTable pivotTable = worksheet.PivotTables["PivotTable1"]; // Remove the pivot table from the collection. worksheet.PivotTables.Remove(pivotTable);</pre>	
Visual Basic	

```
(PivotTableActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Remove the pivot table from the collection.
worksheet.PivotTables.Remove(pivotTable)
```

How to: Change the PivotTable Layout

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Change the PivotTable Layout](#)

To change the layout form of a PivotTable report, pass the appropriate PivotReportLayout enumeration member to the PivotLayout.SetReportLayout method.

The table below describes the available report layouts and lists additional options that can be set for each layout form.

Layout Form	Description	Additional Layout Options																												
Compact Form (PivotReportLayout.Compact)	<p>This is the default layout for a newly created PivotTable report. This layout enables you to keep your report as narrow as possible. It displays items from different row fields in a single column and subsidiary row field items are shown indented to the left.</p> <div><table><tr><th>Row Labels</th><th>Sum of Sales</th></tr><tr><td>☐ Dairy Products</td><td>\$52,562.00</td></tr><tr><td> Camembert Pierrot</td><td>\$13,190.00</td></tr><tr><td> Gorgonzola Telino</td><td>\$6,190.00</td></tr><tr><td> Mascarpone Fabioli</td><td>\$11,833.00</td></tr><tr><td> Mozzarella di Giovanni</td><td>\$6,974.00</td></tr><tr><td> Raclette Courdavault</td><td>\$14,375.00</td></tr><tr><td>☐ Grains/Cereals</td><td>\$39,924.00</td></tr><tr><td> Gnocchi di nonna Alice</td><td>\$5,976.00</td></tr><tr><td> Gustaf's Knäckebröd</td><td>\$10,465.00</td></tr><tr><td> Ravioli Angelo</td><td>\$11,390.00</td></tr><tr><td> Singaporean Hokkien Fried Mee</td><td>\$6,976.00</td></tr><tr><td> Wimmers gute Semmelknödel</td><td>\$5,117.00</td></tr><tr><td>Grand Total</td><td>\$92,486.00</td></tr></table></div>	Row Labels	Sum of Sales	☐ Dairy Products	\$52,562.00	Camembert Pierrot	\$13,190.00	Gorgonzola Telino	\$6,190.00	Mascarpone Fabioli	\$11,833.00	Mozzarella di Giovanni	\$6,974.00	Raclette Courdavault	\$14,375.00	☐ Grains/Cereals	\$39,924.00	Gnocchi di nonna Alice	\$5,976.00	Gustaf's Knäckebröd	\$10,465.00	Ravioli Angelo	\$11,390.00	Singaporean Hokkien Fried Mee	\$6,976.00	Wimmers gute Semmelknödel	\$5,117.00	Grand Total	\$92,486.00	<ul style="list-style-type: none">• PivotLayout.IndentInCompactForm Allows you to specify the amount by which items from different row fields are indented.• PivotLayout.ShowAllSubtotals Allows you to displays all subtotals in a pivot table and specify their location for the outer row fields.• PivotLayout.HideAllSubtotals Allows you to hide all subtotals in a pivot table.• PivotLayout.ShowColumnGrandTotals Allows you to display or hide grand totals for columns.• PivotLayout.ShowRowGrandTotals Allows you to display or hide
Row Labels	Sum of Sales																													
☐ Dairy Products	\$52,562.00																													
Camembert Pierrot	\$13,190.00																													
Gorgonzola Telino	\$6,190.00																													
Mascarpone Fabioli	\$11,833.00																													
Mozzarella di Giovanni	\$6,974.00																													
Raclette Courdavault	\$14,375.00																													
☐ Grains/Cereals	\$39,924.00																													
Gnocchi di nonna Alice	\$5,976.00																													
Gustaf's Knäckebröd	\$10,465.00																													
Ravioli Angelo	\$11,390.00																													
Singaporean Hokkien Fried Mee	\$6,976.00																													
Wimmers gute Semmelknödel	\$5,117.00																													
Grand Total	\$92,486.00																													

		<p>grand totals for rows.</p> <ul style="list-style-type: none">PivotLayout.InsertBlankRows Allows you to display a blank row after items of the outer row fields in a pivot table.
--	--	---

Outline Form
(PivotReportLayout.Outline)

This layout displays each row field in its own column. Subsidiary row field items begin one row below the parent field item.

Category	Product	Sum of Sales
Dairy Products		\$52,562.00
	Camembert Pierrot	\$13,190.00
	Gorgonzola Telino	\$6,190.00
	Mascarpone Fabioli	\$11,833.00
	Mozzarella di Giovanni	\$6,974.00
	Raclette Courdavault	\$14,375.00
Grains/Cereals		\$39,924.00
	Gnocchi di nonna Alice	\$5,976.00
	Gustaf's Knäckebröd	\$10,465.00
	Ravioli Angelo	\$11,390.00
	Singaporean Hokkien Fried Mee	\$6,976.00
	Wimmers gute Semmelknödel	\$5,117.00
Grand Total		\$92,486.00

- PivotLayout.RepeatAllItemLabels
Allows you to repeat item labels for all outer row and column fields in a pivot table.
- PivotLayout.ShowAllSubtotals
Allows you to displays all subtotals in a pivot table and specify their location for the outer row fields.
- PivotLayout.HideAllSubtotals
Allows you to hide all subtotals in a pivot table.
- PivotLayout.ShowColumnGrandTotals
Allows you to display or hide grand totals for columns.
- PivotLayout.ShowRowGrandTotals
Allows you to display

or hide grand totals for rows.

- PivotLayout.InsertBlankRows
Allows you to display a blank row after items of the outer row fields in a pivot table.

Tabular Form
(PivotReportLayout.Tabular)

This layout displays each row field in its own column. Subsidiary row field items begin on the same row. Subtotals for items in the outer row fields are always shown at the bottom.

Category	Product	Sum of Sales
Dairy Products	Camembert Pierrot	\$13,190.00
	Gorgonzola Telino	\$6,190.00
	Mascarpone Fabioli	\$11,833.00
	Mozzarella di Giovanni	\$6,974.00
	Raclette Courdavault	\$14,375.00
Dairy Products Total		\$52,562.00
Grains/Cereals	Gnocchi di nonna Alice	\$5,976.00
	Gustaf's Knäckebröd	\$10,465.00
	Ravioli Angelo	\$11,390.00
	Singaporean Hokkien Fried Mee	\$6,976.00
	Wimmers gute Semmelknödel	\$5,117.00
Grains/Cereals Total		\$39,924.00
Grand Total		\$92,486.00

- PivotLayout.RepeatAllItemLabels
Allows you to repeat item labels for all outer row and column fields in a pivot table.
- PivotLayout.MergeTitles
Allows you to merge and center cells containing item labels for the outer row and column fields, subtotal and grand total labels.
- PivotLayout.HideAllSubtotals
Allows you to hide all subtotals in a pivot table.
- PivotLayout.ShowColumnGrandTotals
Allows you to display or hide grand totals for columns.

		<ul style="list-style-type: none">• PivotLayout .ShowRowGrandTotals Allows you to display or hide grand totals for rows.• PivotLayout .InsertBlankRows Allows you to display a blank row after items of the outer row fields in a pivot table.
--	--	---

The following code example shows how to change the default PivotTable layout to outline form.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

(PivotTableLayoutActions.cs)

```
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Display the pivot table in the outline form.
pivotTable.Layout.SetReportLayout(PivotReportLayout.Outline);
```

Visual Basic

(PivotTableLayoutActions.vb)

```
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Display the pivot table in the outline form.
pivotTable.Layout.SetReportLayout(PivotReportLayout.Outline)
```

Note

Besides changing the layout of the entire pivot table, you can also change the layout of a specific field in the report. To do this, use the corresponding properties of the PivotFieldLayout object, which can be accessed using the field's PivotField.Layout property.

How to: Subtotal Fields in a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Subtotal Fields in a Pivot Table](#)

This topic describes how to manage subtotals in a pivot table. Subtotals automatically appear for [outer fields](#) when you add multiple row and/or column fields to your report.

Row Labels	Sum of AMOUNT
Dairy Products	\$ 201,826.00
Camembert Pierrot	\$ 63,392.00
Mascarpone Fabioli	\$ 62,246.00
Mozzarella di Giovanni	\$ 76,188.00
Grains/Cereals	\$ 127,387.00
Gnocchi di nonna Alice	\$ 29,692.00
Gustaf's Knäckebröd	\$ 19,159.00
Singaporean Hokkien Fried Mee	\$ 45,675.00
Wimmers gute Semmelknödel	\$ 32,861.00
Grand Total	\$ 329,213.00

Subtotals

Select the task you wish to perform.

- [Show or Hide Subtotals For a Pivot Table](#)
- [Modify Subtotals for a PivotTable Field](#)

Show or Hide Subtotals For a Pivot Table

To show or hide subtotals in a PivotTable report, use the following methods of the PivotLayout object accessible using the PivotTable.Layout property.

Method	Description
PivotLayout.ShowAllSubtotals	Displays all subtotals in a pivot table. The method's <i>topOfGroup</i> parameter specifies the subtotal location for the outer row fields in compact or outline form.
PivotLayout.HideAllSubtotals	Hides all subtotals in a pivot table.

The following code example displays all subtotals in a pivot table. Subtotals for the "Category" row field are shown at the bottom of each item in the field.

C#

```
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Show all subtotals at the bottom of each group.
pivotTable.Layout.ShowAllSubtotals(false);
```

Visual Basic

```
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Show all subtotals at the bottom of each group.
pivotTable.Layout.ShowAllSubtotals(False)
```

The image below shows the resulting report (the workbook is opened in Microsoft® Excel®).

Row Labels	Sum of Sales
Dairy Products	
Camembert Pierrot	\$ 13,190.00
Gorgonzola Telino	\$ 6,190.00
Mascarpone Fabioli	\$ 11,833.00
Mozzarella di Giovanni	\$ 6,974.00
Raclette Courdavault	\$ 14,375.00
Dairy Products Total	\$ 52,562.00
Grains/Cereals	
Gnocchi di nonna Alice	\$ 5,976.00
Gustaf's Knäckebröd	\$ 10,465.00
Ravioli Angelo	\$ 11,390.00
Singaporean Hokkien Fried Mee	\$ 6,976.00
Wimmers gute Semmelknödel	\$ 5,117.00
Grains/Cereals Total	\$ 39,924.00
Grand Total	\$ 92,486.00

Modify Subtotals for a PivotTable Field

The table below describes properties and methods you can use to specify and adjust subtotals for a specific [row](#) or [column field](#) in a report.

Member	Description
PivotField.SetSubtotalAutomatic	Displays automatic subtotals for a given field.
PivotField.SetSubtotal	Allows you to change the default summary calculation or to show multiple subtotals for a given field. To remove subtotals for a field, pass the PivotSubtotalFunctions.None value to the method as a parameter.
PivotSubtotalFunctions	Specifies summary functions used to calculate subtotals for a PivotTable field.
PivotFieldLayout.SubtotalOnTop	Specifies the subtotal location for an outer row field shown in outline or compact form. You can display subtotals at the top or bottom of the field's items.
PivotField.SubtotalCaption	Specifies the text to be displayed in the field's subtotal row or column heading.

The following example demonstrates how to use multiple functions to subtotal the "Category" row field. To do this, combine the required [PivotSubtotalFunctions](#) enumeration values with a bitwise OR operator.

Note

You cannot change the subtotal function for a field containing a [calculated item](#). In this case, a **System.InvalidOperationException** exception will be thrown.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotFieldActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the pivot field by its name in the collection.
PivotField field = pivotTable.Fields["Category"];
// Display multiple subtotals for the field.
field.SetSubtotal(PivotSubtotalFunctions.Sum | PivotSubtotalFunctions.Average);
```

Visual Basic

```
(PivotFieldActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the pivot field by its name in the collection.
Dim field As PivotField = pivotTable.Fields("Category")
' Display multiple subtotals for the field.
field.SetSubtotal(PivotSubtotalFunctions.Sum Or PivotSubtotalFunctions.Average)
```

The image below shows the resulting report (the workbook is opened in Microsoft® Excel®).

Row Labels	Sum of Sales
Dairy Products	
Camembert Pierrot	\$ 13,190.00
Gorgonzola Telino	\$ 6,190.00
Mascarpone Fabioli	\$ 11,833.00
Mozzarella di Giovanni	\$ 6,974.00
Raclette Courdavault	\$ 14,375.00
Dairy Products Sum	\$ 52,562.00
Dairy Products Average	\$ 2,628.10
Grains/Cereals	
Gnocchi di nonna Alice	\$ 5,976.00
Gustaf's Knäckebröd	\$ 10,465.00
Ravioli Angelo	\$ 11,390.00
Singaporean Hokkien Fried Mee	\$ 6,976.00
Wimmers gute Semmelknödel	\$ 5,117.00
Grains/Cereals Sum	\$ 39,924.00
Grains/Cereals Average	\$ 1,996.20
Grand Total	\$ 92,486.00

How to: Display or Hide Grand Totals for a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Display or Hide Grand Totals for a Pivot Table](#)

To control how grand totals are displayed in a pivot table, use the following properties.

Property	Description
PivotLayout.ShowRowGrandTotals	Specifies whether to display a grand total column.
PivotLayout.ShowColumnGrandTotals	Specifies whether to display a grand total row.
PivotViewOptions.GrandTotalCaption	Specifies the text label for both the grand total column and grand total row.

Hide Grand Totals for Rows

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableLayoutActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Add the "Region" field to the column axis area.
pivotTable.ColumnFields.Add(pivotTable.Fields["Region"]);
// Hide grand totals for rows.
pivotTable.Layout.ShowRowGrandTotals = false;
```

Visual Basic

```
(PivotTableLayoutActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Add the "Region" field to the column axis area.
pivotTable.ColumnFields.Add(pivotTable.Fields("Region"))
' Hide grand totals for rows.
pivotTable.Layout.ShowRowGrandTotals = False
```

The image below shows the resulting PivotTable report (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F
1						
2	Sum of Sales	Column Labels ▾				
3	Row Labels ▾	Midwest	Northeast	South	West	
4	Dairy Products	\$ 13,195.00	\$ 11,997.00	\$ 13,650.00	\$ 13,720.00	
5	Camembert Pierrot	\$ 2,820.00	\$ 3,425.00	\$ 3,425.00	\$ 3,520.00	
6	Gorgonzola Telino	\$ 1,725.00	\$ 1,200.00	\$ 1,765.00	\$ 1,500.00	
7	Mascarpone Fabioli	\$ 3,000.00	\$ 2,448.00	\$ 3,260.00	\$ 3,125.00	
8	Mozzarella di Giovanni	\$ 2,150.00	\$ 1,044.00	\$ 1,955.00	\$ 1,825.00	
9	Raclette Courdavault	\$ 3,500.00	\$ 3,880.00	\$ 3,245.00	\$ 3,750.00	
10	Grains/Cereals	\$ 10,705.00	\$ 8,739.00	\$ 10,035.00	\$ 10,445.00	
11	Gnocchi di nonna Alice	\$ 1,650.00	\$ 1,216.00	\$ 1,435.00	\$ 1,675.00	
12	Gustaf's Knäckebröd	\$ 2,980.00	\$ 2,420.00	\$ 2,345.00	\$ 2,720.00	
13	Ravioli Angelo	\$ 2,985.00	\$ 2,390.00	\$ 2,965.00	\$ 3,050.00	
14	Singaporean Hokkien Fried Mee	\$ 1,825.00	\$ 1,616.00	\$ 1,835.00	\$ 1,700.00	
15	Wimmers gute Semmelknödel	\$ 1,265.00	\$ 1,097.00	\$ 1,455.00	\$ 1,300.00	
16	Grand Total	\$ 23,900.00	\$ 20,736.00	\$ 23,685.00	\$ 24,165.00	
17						

Report1 Blank +

Hide Grand Totals for Columns

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableLayoutActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Add the "Region" field to the column axis area.
pivotTable.ColumnFields.Add(pivotTable.Fields["Region"]);
// Hide grand totals for columns.
pivotTable.Layout.ShowColumnGrandTotals = false;
```

Visual Basic

```
(PivotTableLayoutActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Add the "Region" field to the column axis area.
pivotTable.ColumnFields.Add(pivotTable.Fields("Region"))
' Hide grand totals for columns.
pivotTable.Layout.ShowColumnGrandTotals = False
```

The image below shows the resulting PivotTable report (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F
1						
2	Sum of Sales	Column Labels ▼				
3	Row Labels ▼	Midwest	Northeast	South	West	Grand Total
4	☐ Dairy Products	\$ 13,195.00	\$ 11,997.00	\$ 13,650.00	\$ 13,720.00	\$ 52,562.00
5	Camembert Pierrot	\$ 2,820.00	\$ 3,425.00	\$ 3,425.00	\$ 3,520.00	\$ 13,190.00
6	Gorgonzola Telino	\$ 1,725.00	\$ 1,200.00	\$ 1,765.00	\$ 1,500.00	\$ 6,190.00
7	Mascarpone Fabioli	\$ 3,000.00	\$ 2,448.00	\$ 3,260.00	\$ 3,125.00	\$ 11,833.00
8	Mozzarella di Giovanni	\$ 2,150.00	\$ 1,044.00	\$ 1,955.00	\$ 1,825.00	\$ 6,974.00
9	Raclette Courdavault	\$ 3,500.00	\$ 3,880.00	\$ 3,245.00	\$ 3,750.00	\$ 14,375.00
10	☐ Grains/Cereals	\$ 10,705.00	\$ 8,739.00	\$ 10,035.00	\$ 10,445.00	\$ 39,924.00
11	Gnocchi di nonna Alice	\$ 1,650.00	\$ 1,216.00	\$ 1,435.00	\$ 1,675.00	\$ 5,976.00
12	Gustaf's Knäckebröd	\$ 2,980.00	\$ 2,420.00	\$ 2,345.00	\$ 2,720.00	\$ 10,465.00
13	Ravioli Angelo	\$ 2,985.00	\$ 2,390.00	\$ 2,965.00	\$ 3,050.00	\$ 11,390.00
14	Singaporean Hokkien Fried Mee	\$ 1,825.00	\$ 1,616.00	\$ 1,835.00	\$ 1,700.00	\$ 6,976.00
15	Wimmers gute Semmelknödel	\$ 1,265.00	\$ 1,097.00	\$ 1,455.00	\$ 1,300.00	\$ 5,117.00
16						

Report1 Blank (+)

How to: Apply a Predefined Style to a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Apply a Predefined Style to a Pivot Table](#)

The example below demonstrates how to change the appearance of the existing pivot table using one of the predefined pivot table styles. To apply a style, assign the corresponding `TableStyle` object to the `PivotTable.Style` property. Access the required pivot table style from the `IWorkbook.TableStyles` collection by its ID (a member of the **BuiltInPivotStyleId** enumeration).

Note that you can also create your own style and apply it to a pivot table, as shown in the [How to: Apply a Custom Style to a Pivot Table](#) example.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableFormattingActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Set the pivot table style.
pivotTable.Style = workbook.TableStyles[BuiltInPivotStyleId.PivotStyleDark7];
```

Visual Basic

```
(PivotTableFormattingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Set the pivot table style.
pivotTable.Style = workbook.TableStyles(BuiltInPivotStyleId.PivotStyleDark7)
```

The image below shows the pivot table appearance when the *PivotStyleDark7* style is applied (the workbook is opened in Microsoft® Excel®).

	A	B	C	D
1				
2		Row Labels	Sum of Sales	
3		Dairy Products	\$ 52,562.00	
4		Camembert Pierrot	\$ 13,190.00	
5		Gorgonzola Telino	\$ 6,190.00	
6		Mascarpone Fabioli	\$ 11,833.00	
7		Mozzarella di Giovanni	\$ 6,974.00	
8		Raclette Courdavault	\$ 14,375.00	
9		Grains/Cereals	\$ 39,924.00	
10		Gnocchi di nonna Alice	\$ 5,976.00	
11		Gustaf's Knäckebröd	\$ 10,465.00	
12		Ravioli Angelo	\$ 11,390.00	
13		Singaporean Hokkien Fried Mee	\$ 6,976.00	
14		Wimmers gute Semmelknödel	\$ 5,117.00	
15		Grand Total	\$ 92,486.00	
16				

Report1

BI ...

See Also
[How to: Control Style Options](#)
[How to: Apply a Custom Style to a Pivot Table](#)

How to: Apply a Custom Style to a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Apply a Custom Style to a Pivot Table](#)

This example demonstrates how to create a custom style and apply it to a pivot table. By default, the workbook's collection of pivot table styles (TableStyleCollection) contains built-in styles similar to Microsoft® Excel® and the *None* style, which specifies that no formatting should be applied to the pivot table.

A pivot table style is defined by the TableStyle object, which consists of a collection of table style elements (TableStyle.TableStyleElements). Each table style element (TableStyleElement) specifies formatting for a particular element of a pivot table. The TableStyleElementType enumerator lists the supported table style elements. Use properties of the TableStyleElement object to customize borders (TableStyleElement.Borders), fill (TableStyleElement.Fill) and font (TableStyleElement.Font) for the corresponding element of a pivot table.

To create a custom pivot table style, do the following.

1. Add a new pivot table style to the IWorkbook.TableStyles collection by calling the TableStyleCollection.Add method. This method returns the TableStyle object that represents the newly created pivot table style.
- Note that you can also create a custom pivot table style based on the existing pivot table style (e.g., a built-in pivot table style). To do this, use the TableStyle.Duplicate method. This method creates a copy of the specified style and returns the TableStyle object representing the created style.
2. Set the TableStyle.IsPivotStyle property to **true** and the TableStyle.IsTableStyle property to **false** to indicate that the created style should be applied only to pivot tables. For the custom styles that are based on the built-in pivot table styles, this step can be skipped since such styles copy appropriate values of these properties from the built-in styles.
3. Call the TableStyle.BeginUpdate method.
4. Access the table style element to be modified from the TableStyle.TableStyleElements collection by the corresponding TableStyleElementType enumeration member. Use the TableStyleElement properties to specify the required formatting for the element. If you need to remove existing formatting from the element, use its TableStyleElement.Clear method.

Repeat this step for all table style elements you wish to modify.

5. Call the TableStyle.EndUpdate method.
6. Apply the created style to the pivot table by using the PivotTable.Style property.

For more details on how to manage the workbook's collection of table and pivot table styles, refer to the [How to: Create, Modify And Delete Table Styles](#) topic.

The example below duplicates the built-in pivot table style and modifies the new style by changing formatting characteristics for the entire table, column headers and the grand total row.

C#	
----	--

```
// Access the pivot table by its name in the collection
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Get the pivot table style to be duplicated.
TableStyle sourceStyle = workbook.TableStyles[BuiltInPivotStyleId.PivotStyleMedium3];
// Duplicate the pivot table style.
TableStyle customStyle = sourceStyle.Duplicate();
// Modify the required formatting characteristics of the created pivot table style.
customStyle.BeginUpdate();
try
{
    // Specify formatting characteristics for the column headers.
    TableStyleElement header = customStyle.TableStyleElements[TableStyleElementType.HeaderRow];
    header.Fill.BackgroundColor = Color.FromArgb(0x1F, 0x3E, 0x7E);
    // Specify formatting characteristics for the whole table.
    TableStyleElement wholeTable = customStyle.TableStyleElements[TableStyleElementType.WholeTable];
    wholeTable.Fill.BackgroundColor = Color.FromArgb(0xF1, 0xF4, 0xD0);
    wholeTable.Borders.RemoveBorders();
    // Specify formatting characteristics for the grand total row.
    TableStyleElement totalRowStyle = customStyle.TableStyleElements[TableStyleElementType.TotalRow];
    totalRowStyle.Fill.BackgroundColor = Color.FromArgb(166, 166, 166);
    totalRowStyle.Borders.RemoveBorders();
}
finally
{
    customStyle.EndUpdate();
}
// Apply the created custom style to the pivot table.
pivotTable.Style = customStyle;
```

Visual Basic

```
' Access the pivot table by its name in the collection
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Get the pivot table style to be duplicated.
Dim sourceStyle As TableStyle = workbook.TableStyles(BuiltInPivotStyleId.PivotStyleMedium3)
' Duplicate the pivot table style.
Dim customStyle As TableStyle = sourceStyle.Duplicate()
' Modify the required formatting characteristics of the created pivot table style.
customStyle.BeginUpdate()
Try
    ' Specify formatting characteristics for the column headers.
    Dim header As TableStyleElement = customStyle.TableStyleElements(TableStyleElementType.HeaderRow)
    header.Fill.BackgroundColor = Color.FromArgb(&H1F, &H3E, &H7E)
    ' Specify formatting characteristics for the whole table.
    Dim wholeTable As TableStyleElement = customStyle.TableStyleElements(TableStyleElementType.WholeTable)
    wholeTable.Fill.BackgroundColor = Color.FromArgb(&HF1, &HF4, &HD0)
    wholeTable.Borders.RemoveBorders()
    ' Specify formatting characteristics for the grand total row.
    Dim totalRowStyle As TableStyleElement = customStyle.TableStyleElements(TableStyleElementType.TotalRow)
    totalRowStyle.Fill.BackgroundColor = Color.FromArgb(166, 166, 166)
    totalRowStyle.Borders.RemoveBorders()
Finally
    customStyle.EndUpdate()
End Try
' Apply the created custom style to the pivot table.
pivotTable.Style = customStyle
```

The image below illustrates the pivot table appearance when the custom style is applied (the workbook is opened in Microsoft® Excel®).

	A	B	C	D
1				
2		Row Labels	Sum of Sales	
3		Dairy Products	\$ 52,562.00	
4		Camembert Pierrot	\$ 13,190.00	
5		Gorgonzola Telino	\$ 6,190.00	
6		Mascarpone Fabioli	\$ 11,833.00	
7		Mozzarella di Giovanni	\$ 6,974.00	
8		Raclette Courdavault	\$ 14,375.00	
9		Grains/Cereals	\$ 39,924.00	
10		Gnocchi di nonna Alice	\$ 5,976.00	
11		Gustaf's Knäckebröd	\$ 10,465.00	
12		Ravioli Angelo	\$ 11,390.00	
13		Singaporean Hokkien Fried Mee	\$ 6,976.00	
14		Wimmers gute Semmelknödel	\$ 5,117.00	
15		Grand Total	\$ 92,486.00	
16				

Report

See Also
[How to: Apply a Predefined Style to a Pivot Table](#)
[How to: Control Style Options](#)

How to: Control Style Options

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Control Style Options](#)

A [pivot table style](#) consists of a collection of table style elements (TableStyleElement), each of which specifies formatting for a particular element of a pivot table. The TableStyleElementType enumerator lists supported table style element types. The PivotTable object provides specific properties that allow you to configure PivotTable style options by specifying table elements to be formatted as defined by the corresponding elements of the applied pivot table style.

• Format Row Headers

Use the PivotTable.ShowRowHeaders property to enable or disable style formatting for row headers in a pivot table. The appearance of row headers is specified by the TableStyleElementType.FirstRowSubheading, TableStyleElementType.SecondRowSubheading and TableStyleElementType.ThirdRowSubheading elements of the applied pivot table style.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableFormattingActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Add the "Region" field to the column axis area.
pivotTable.ColumnFields.Add(pivotTable.Fields["Region"]);
// Remove formatting from row headers.
pivotTable.ShowRowHeaders = false;
```

Visual Basic

```
(PivotTableFormattingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Add the "Region" field to the column axis area.
pivotTable.ColumnFields.Add(pivotTable.Fields("Region"))
' Remove formatting from row headers.
pivotTable.ShowRowHeaders = False
```

Format Column Headers

Use the PivotTable.ShowColumnHeaders property to enable or disable style formatting for column headers in a pivot table. The appearance of column headers is specified by the TableStyleElementType.HeaderRow, TableStyleElementType.FirstColumnSubheading, TableStyleElementType.SecondColumnSubheading and TableStyleElementType.ThirdColumnSubheading elements of the applied pivot table style.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableFormattingActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Remove formatting from column headers.
pivotTable.ShowColumnHeaders = false;
```

Visual Basic

```
(PivotTableFormattingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Remove formatting from column headers.
pivotTable.ShowColumnHeaders = False
```

Apply Banded Row Formatting

Set the `PivotTable.BandedRows` property to **true** to shade alternate rows in a pivot table as specified by the `TableStyleElementType.FirstRowStripe` and `TableStyleElementType.SecondRowStripe` elements of the applied pivot table style. To specify a number of `PivotTable` rows to be included into odd and even row stripes, use the `TableStyleElement.StripeSize` property of these table style elements.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableFormattingActions.cs)
Worksheet worksheet = workbook.Worksheets["Report4"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Apply the banded row formatting to the pivot table.
pivotTable.BandedRows = true;
```

Visual Basic

```
(PivotTableFormattingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report4")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Apply the banded row formatting to the pivot table.
pivotTable.BandedRows = True
```

Apply Banded Column Formatting

Set the `PivotTable.BandedColumns` property to **true** to shade alternate columns in a pivot table as specified by the `TableStyleElementType.FirstColumnStripe` and `TableStyleElementType.SecondColumnStripe` elements of the applied pivot table style. To specify the number of `PivotTable` columns to be displayed as odd and even column stripes, use the `TableStyleElement.StripeSize` property of these table style elements.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableFormattingActions.cs)
Worksheet worksheet = workbook.Worksheets["Report4"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Apply the banded column formatting to the pivot table.
pivotTable.BandedColumns = true;
```

Visual Basic

```
(PivotTableFormattingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report4")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Apply the banded column formatting to the pivot table.
pivotTable.BandedColumns = True
```

How to: Change the Summary Function for a Data Field

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Change the Summary Function for a Data Field](#)

To change the default summary function applied to a [data field](#) in the PivotTable report, assign the required PivotDataConsolidationFunction enumeration value to the PivotDataField.SummarizeValuesBy property.

For example, the following code demonstrates how to summarize values in a data field using the "Average" built-in function.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(ValueFieldSettingsActions.cs)
Worksheet sourceWorksheet = workbook.Worksheets["Data5"];
Worksheet worksheet = workbook.Worksheets.Add();
workbook.Worksheets.ActiveWorksheet = worksheet;
// Create a pivot table using the cell range "A1:E65" as the data source.
PivotTable pivotTable = worksheet.PivotTables.Add(sourceWorksheet["A1:E65"], worksheet["B2"]);
// Add the "Category" field to the row axis area.
pivotTable.RowFields.Add(pivotTable.Fields["Category"]);
// Add the "Product" field to the row axis area.
pivotTable.RowFields.Add(pivotTable.Fields["Product"]);
// Add the "Amount" field to the data area.
PivotDataField dataField = pivotTable.DataFields.Add(pivotTable.Fields["Amount"]);
// Use the "Average" function to summarize values in the data field.
dataField.SummarizeValuesBy = PivotDataConsolidationFunction.Average;
// Specify the number format for the data field.
dataField.NumberFormat = "@" ([$$-409] * #,##0.00 ); ([$$-409] * (#,##0.00); ([$$-409] * "" - "?? "); (@ )";
```

Visual Basic

```
(ValueFieldSettingsActions.vb)
Dim sourceWorksheet As Worksheet = workbook.Worksheets("Data5")
Dim worksheet As Worksheet = workbook.Worksheets.Add()
workbook.Worksheets.ActiveWorksheet = worksheet
' Create a pivot table using the cell range "A1:E65" as the data source.
Dim pivotTable As PivotTable = worksheet.PivotTables.Add(sourceWorksheet("
' Add the "Category" field to the row axis area.
pivotTable.RowFields.Add(pivotTable.Fields("Category"))
' Add the "Product" field to the row axis area.
pivotTable.RowFields.Add(pivotTable.Fields("Product"))
' Add the "Amount" field to the data area.
Dim dataField As PivotDataField = pivotTable.DataFields.Add(pivotTable.Fie
' Use the "Average" function to summarize values in the data field.
dataField.SummarizeValuesBy = PivotDataConsolidationFunction.Average
' Specify the number format for the data field.
dataField.NumberFormat = " ([$$-409] * #,##0.00 ); ([$$-409] * (#,##0.00); (
```

The image below shows the resulting PivotTable report (the workbook is opened in Microsoft® Excel®).

Row Labels		Average of AMOUNT
☐ Dairy Products	\$	5,766.46
Camembert Pierrot	\$	5,282.67
Mascarpone Fabioli	\$	6,224.60
Mozzarella di Giovanni	\$	5,860.62
☐ Grains/Cereals	\$	4,392.66
Gnocchi di nonna Alice	\$	4,948.67
Gustaf's Knäckebröd	\$	3,831.80
Singaporean Hokkien Fried Mee	\$	4,567.50
Wimmers gute Semmelknödel	\$	4,107.63
Grand Total	\$	5,143.95

How to: Apply a Custom Calculation to a Data Field

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Apply a Custom Calculation to a Data Field](#)

Summaries in the [data area](#) are calculated according to the data field's `PivotDataField.SummarizeValuesBy` property. By default, they are displayed "as is". However, you can use the `PivotDataField.ShowValuesWithCalculation` method overloads to apply an additional calculation option to the raw summary values to change the way they are displayed in the PivotTable report.

The available calculation options are defined by the values of the `PivotShowValuesAsType` enumeration and listed in the table below. Depending on the selected option, you may need to specify either a *base field* or a *base item* and its *base item* that will serve as a basis for a custom calculation. Only a field from the [row](#) or [column area](#) can be used as a base field, otherwise, all results will be shown as a **#N/A** error.

Calculation Option	PivotShowValuesAsType Member	Description
% of Grand Total	<code>PivotShowValuesAsType.PercentOfTotal</code>	Displays values as the percentage of the grand total of all values in the PivotTable report.
% of Column Total	<code>PivotShowValuesAsType.PercentOfColumn</code>	Displays values in each column as the percentage of the total value of the column.
% of Row Total	<code>PivotShowValuesAsType.PercentOfRow</code>	Displays values in each row as the percentage of the total value of the row.
% of Parent Column Total	<code>PivotShowValuesAsType.PercentOfParentColumn</code>	Displays values in each inner column as the percentage of the total of the parent item in the outer column.
% of Parent Row Total	<code>PivotShowValuesAsType.PercentOfParentRow</code>	Displays values in each inner row as the percentage of the total of the parent item in the outer row.
Index	<code>PivotShowValuesAsType.Index</code>	Calculates the relative importance of each summary value by using the following formula: $((\text{value in cell}) \times (\text{Grand Total of Grand Totals})) / ((\text{Grand Row Total}) \times (\text{Grand Column Total}))$
% of Parent Total	<code>PivotShowValuesAsType.PercentOfParent</code>	Displays values in a data field as the percentage of the value of the parent item in the specified <i>base field</i> .
Rank Smallest to Largest	<code>PivotShowValuesAsType.RankAscending</code>	Displays the rank of values for a specific <i>base field</i> , listing the smallest item in the field as 1, and each larger value with a higher rank value.
Rank Largest to Smallest	<code>PivotShowValuesAsType.RankDescending</code>	Displays the rank of values for a specific <i>base field</i> , listing the largest item in the field as 1, and each smaller value with a higher rank value.
Running Total In	<code>PivotShowValuesAsType.RunningTotal</code>	Displays values for successive items in the <i>base field</i> as a running total.
% Running Total In	<code>PivotShowValuesAsType.PercentOfRunningTotal</code>	Calculates a running total for successive items in the <i>base field</i> as a percentage of the grand total value.
% Of	<code>PivotShowValuesAsType.Percent</code>	Displays values as the percentage of the value of the <i>base item</i> in the <i>base field</i> .

Difference From	PivotShowValuesAsType.Difference	Displays values as the difference from the value of the <i>base item</i> in the <i>base field</i> .
% Difference From	PivotShowValuesAsType.PercentDifference	Displays values as the percentage difference of the value of the <i>base item</i> in the <i>base field</i> .

To remove any custom calculation applied to a data field, pass the `PivotShowValuesAsType.NoCalculation` enumeration member to the `PivotDataField.ShowValuesWithCalculation` method as a parameter, or call the `PivotDataField.ShowValuesWithoutCalculation` method.

The following example demonstrates how to calculate the difference in product sales between the current quarter and the previous quarter. Values in the data field are summarized using the "Sum" function.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(ValueFieldSettingsActions.cs)
Worksheet worksheet = workbook.Worksheets["Report14"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the data field by its index in the collection.
PivotDataField dataField = pivotTable.DataFields[0];
// Display the difference in product sales between the current quarter and the previous quarter.
dataField.ShowValuesWithCalculation(PivotShowValuesAsType.Difference, pivotTable.Fields["Quarter"], PivotTable1
```

Visual Basic

```
(ValueFieldSettingsActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report14")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the data field by its index in the collection.
Dim dataField As PivotDataField = pivotTable.DataFields(0)
' Display the difference in product sales between the current quarter and
dataField.ShowValuesWithCalculation(PivotShowValuesAsType.Difference, pivoc
```

The image below shows the result.

No Calculation

Sum of AMOUNT		Column Labels			
Row Labels		Q1	Q2	Q3	Q4
Dairy Products		\$ 46,070.00	\$ 53,979.00	\$ 41,290.00	\$ 60,487.00
Camembert Pierrot		\$ 12,424.00	\$ 15,996.00	\$ 15,164.00	\$ 19,808.00
Mascarpone Fabioli		\$ 20,794.00	\$ 12,240.00	\$ 8,266.00	\$ 20,946.00
Mozzarella di Giovanni		\$ 12,852.00	\$ 25,743.00	\$ 17,860.00	\$ 19,733.00
Grains/Cereals		\$ 21,179.00	\$ 27,004.00	\$ 40,504.00	\$ 38,700.00
Gnocchi di nonna Alice		\$ 9,475.00	\$ 2,592.00	\$ 4,125.00	\$ 13,500.00
Gustaf's Knäckebröd		\$ 1,768.00	\$ 4,735.00	\$ 4,098.00	\$ 8,558.00
Singaporean Hokkien Fried Mee		\$ 8,050.00	\$ 13,152.00	\$ 18,261.00	\$ 6,212.00
Wimmers gute Semmelknödel		\$ 1,886.00	\$ 6,525.00	\$ 14,020.00	\$ 10,430.00
Grand Total		\$ 67,249.00	\$ 80,983.00	\$ 81,794.00	\$ 99,187.00

Sum of AMOUNT		Column Labels			
Row Labels		Q1	Q2	Q3	Q4
Dairy Products			\$ 7,909.00	\$ (12,689.00)	\$ 19,197.00
Camembert Pierrot			\$ 3,572.00	\$ (832.00)	\$ 4,644.00
Mascarpone Fabioli			\$ (8,554.00)	\$ (3,974.00)	\$ 12,680.00
Mozzarella di Giovanni			\$ 12,891.00	\$ (7,883.00)	\$ 1,873.00
Grains/Cereals			\$ 5,825.00	\$ 13,500.00	\$ (1,804.00)
Gnocchi di nonna Alice			\$ (6,883.00)	\$ 1,533.00	\$ 9,375.00
Gustaf's Knäckebröd			\$ 2,967.00	\$ (637.00)	\$ 4,460.00
Singaporean Hokkien Fried Mee			\$ 5,102.00	\$ 5,109.00	\$ (12,049.00)
Wimmers gute Semmelknödel			\$ 4,639.00	\$ 7,495.00	\$ (3,590.00)
Grand Total			\$ 13,734.00	\$ 811.00	\$ 17,393.00

Difference From Previous Quarter

How to: Create a Calculated Field

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Create a Calculated Field](#)

If the [predefined aggregation functions](#) or [Show Values As](#) calculation options do not meet your requirements, you can create your own formulas to calculate values in a PivotTable report by inserting *calculated fields* and [calculated items](#).

All calculated fields for a pivot table are stored in the PivotCalculatedFieldCollection collection, which can be accessed using the PivotTable.CalculatedFields property. Use the collection's methods to [create](#), [modify](#) or [remove](#) a calculated field.

Create a Calculated Field

To create a calculated field, use the PivotCalculatedFieldCollection.Add method. The first parameter of this method allows you specify a formula for the calculated field.

A formula string should conform to the common syntax rules and contain only supported elements.

- In the formula, you can use constants and refer to other fields in the PivotTable report. The calculation will be performed on the sum of the underlying data for any fields in the formula. When you reference a field in your formula, you can enclose its name in apostrophes or omit them.
- You cannot create formulas that use a cell reference, defined name, circular references and arrays.
- You cannot use worksheet functions that require cell references or defined names as arguments.
- The formula cannot refer to the PivotTable's subtotals, totals and Grand Total value.

After the calculated field is created, add it to the PivotTable's [data area](#) using the PivotDataFieldCollection.Add method of the PivotTable.DataFields collection.

Note

Calculated fields are stored in the PivotCache and available to all pivot tables that share the same cache.

The following code demonstrates how to create a field that calculates a 10% sales tax for values in the "Sales" field.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

(PivotCalculatedFieldActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Create a calculated field based on data in the "Sales" field.
PivotField field = pivotTable.CalculatedFields.Add("=Sales*10%", "Sales Tax");
// Add the calculated field to the data area and specify the custom field name.
PivotDataField dataField = pivotTable.DataFields.Add(field, "Total Tax");
// Specify the number format for the data field.
dataField.NumberFormat = "@" ([\$\$-409]* #,##0.00); ([\$\$-409]* (#,##0.00); ([\$\$-409]* "" - ""??); (@)";

Visual Basic

(PivotCalculatedFieldActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Create a calculated field based on data in the "Sales" field.
Dim field As PivotField = pivotTable.CalculatedFields.Add("=Sales*10%", "Sales Tax")
' Add the calculated field to the data area and specify the custom field name.
Dim dataField As PivotDataField = pivotTable.DataFields.Add(field, "Total Tax")
' Specify the number format for the data field.
dataField.NumberFormat = " ([\$\$-409]* #,##0.00); ([\$\$-409]* (#,##0.00); ([\$\$-409]* "" - ""??); (@)"

The resulting PivotTable report is shown in the image below (the workbook is opened in Microsoft® Excel®).

Row Labels	Sum of Sales	Total Tax
Dairy Products	\$ 52,562.00	\$ 5,256.20
Camembert Pierrot	\$ 13,190.00	\$ 1,319.00
Gorgonzola Telino	\$ 6,190.00	\$ 619.00
Mascarpone Fabioli	\$ 11,833.00	\$ 1,183.30
Mozzarella di Giovanni	\$ 6,974.00	\$ 697.40
Raclette Courdavault	\$ 14,375.00	\$ 1,437.50
Grains/Cereals	\$ 39,924.00	\$ 3,992.40
Gnocchi di nonna Alice	\$ 5,976.00	\$ 597.60
Gustaf's Knäckebröd	\$ 10,465.00	\$ 1,046.50
Ravioli Angelo	\$ 11,390.00	\$ 1,139.00
Singaporean Hokkien Fried Mee	\$ 6,976.00	\$ 697.60
Wimmers gute Semmelknödel	\$ 5,117.00	\$ 511.70
Grand Total	\$ 92,486.00	\$ 9,248.60

Modify a Calculated Field

To change a formula for a calculated field, get access to the required field by its name or index in the PivotCalculatedFieldCollection collection and then assign a new formula to the field's PivotField.Formula property. To rename a calculated field, use the PivotField.Name property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotCalculatedFieldActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Create a calculated field based on data in the "Sales" field.
pivotTable.CalculatedFields.Add("=Sales*10%", "Sales Tax Rate 10");
// Access the calculated field by its name in the collection.
PivotField field = pivotTable.CalculatedFields["Sales Tax Rate 10"];
//Change the formula for the calculated field.
field.Formula = "=Sales*15%";
//Change the calculated field name.
field.Name = "Sales Tax Rate 15";
//Add the calculated field to the data area and specify the custom field name.
PivotDataField dataField = pivotTable.DataFields.Add(field, "Total Tax");
// Specify the number format for the data field.
dataField.NumberFormat = @"([$$-409]* #,##0.00); ([$$-409]* (#,##0.00);
```

Visual Basic

```
(PivotCalculatedFieldActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Create a calculated field based on data in the "Sales" field.
pivotTable.CalculatedFields.Add("=Sales*10%", "Sales Tax Rate 10")
' Access the calculated field by its name in the collection.
Dim field As PivotField = pivotTable.CalculatedFields("Sales Tax Rate 10")
'Change the formula for the calculated field.
field.Formula = "=Sales*15%"
'Change the calculated field name.
field.Name = "Sales Tax Rate 15"
'Add the calculated field to the data area and specify the custom field na
Dim dataField As PivotDataField = pivotTable.DataFields.Add(field, "Total
' Specify the number format for the data field.
dataField.NumberFormat = " ([$$-409]* #,##0.00 ); ([$$-409]* (#,##0.00); (
```

Remove a Calculated Field

To remove a particular calculated field from the collection, use the PivotCalculatedFieldCollection.Remove or PivotCalculatedFieldCollection.RemoveAt method. To remove all calculated fields from the collection at once, use the PivotCalculatedFieldCollection.Clear method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#	
<pre>(PivotCalculatedFieldActions.cs) Worksheet worksheet = workbook.Worksheets["Report1"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Access the pivot table by its name in the collection. PivotTable pivotTable = worksheet.PivotTables["PivotTable1"]; // Create a calculated field based on data in the "Sales" field. pivotTable.CalculatedFields.Add("=Sales*10%", "Sales Tax"); // Access the calculated field by its name in the collection. PivotField field = pivotTable.CalculatedFields["Sales Tax"]; // Add the calculated field to the data area. PivotDataField dataField = pivotTable.DataFields.Add(field); //Remove the calculated field. pivotTable.CalculatedFields.RemoveAt(0);</pre>	
Visual Basic	

```
(PivotCalculatedFieldActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Create a calculated field based on data in the "Sales" field.
pivotTable.CalculatedFields.Add("=Sales*10%", "Sales Tax")
' Access the calculated field by its name in the collection.
Dim field As PivotField = pivotTable.CalculatedFields("Sales Tax")
' Add the calculated field to the data area.
Dim dataField As PivotDataField = pivotTable.DataFields.Add(field)
'Remove the calculated field.
pivotTable.CalculatedFields.RemoveAt(0)
```

How to: Create a Calculated Item

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Create a Calculated Item](#)

If the [predefined aggregation functions](#) or [Show Values As](#) calculation options do not meet your requirements, you can create your own formulas to calculate values in a PivotTable report by inserting [calculated fields](#) and *calculated items*. A calculated item is a custom item in a PivotTable field whose value is produced based on values of other items in the same field.

All calculated items added to a PivotTable field are stored in the PivotCalculatedItemCollection collection, which can be accessed using the PivotField.CalculatedItems property. Use the collection's methods to [create](#), [modify](#) or [remove](#) calculated items.

Calculated Item Limitations

Before inserting a calculated item, take into account the following restrictions.

- You cannot add a calculated item to a [page field](#). Also, you cannot move a [row](#) or [column field](#) containing calculated items to the [page area](#) of the PivotTable report.
- You cannot add a calculated item to a [grouped field](#). Ungroup the field before inserting the calculated item.
- You cannot add multiple copies of a field containing calculated items to the [data area](#).
- You cannot add a calculated item to a PivotTable report that uses the PivotDataConsolidationFunction.Average, PivotDataConsolidationFunction.StdDev, PivotDataConsolidationFunction.StdDevp, PivotDataConsolidationFunction.Var, or PivotDataConsolidationFunction.Varp aggregation function in the [data area](#).

Create a Calculated Item

To create a calculated item, use the PivotCalculatedItemCollection.Add method. The first parameter of this method allows you specify a formula for the calculated item.

A formula string should conform to the common syntax rules and contain only supported elements.

- In the formula, you can use constants and refer to other items in the same field where the calculated item resides. To create an item reference, use one of the approaches listed in the table below. All examples in the table refer to the pivot table shown later in this section.

Refere By	Description	Example
Item Name	Refer to an item by its name in the PivotTable field (PivotItem.Caption). When creating a reference, you can enclose the item's name in apostrophes or omit them.	The following example demonstrates how to sum up the first three items in the "State" field: =Arizona+California+Colorado ...or... ='Arizona'+ 'California'+ 'Colorado'
Item Position	Refer to an item by its position in the PivotTable as currently sorted and displayed. Hidden items are ignored.	The following example demonstrates how to sum up the first three items in the "State" field: =State[1]+State[2]+State[3]
Item Relative Position	Refer to an item by its position relative to the calculated item containing the formula. Positive numbers refer to items below or to the right of the calculated item. Negative numbers refer to items above or to the left of the calculated item. If the specified position is before the first item or after the last item in the field, the #REF! error is displayed.	The following example demonstrates how to sum up the first three items in the "State" field relative to the "West Total" calculated item: =State[-6]+State[-5]+State[-4]

- You cannot create formulas that use a cell reference, defined name, circular references and arrays.
- You cannot use worksheet functions that require cell references or defined names as arguments.
- The formula cannot refer to the PivotTable's subtotals, totals and Grand Total value.

The following code demonstrates how to create two calculated items to calculate total sales for each region.

 [Show Me](#)

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotCalculatedItemActions.cs)
Worksheet worksheet = workbook.Worksheets["Report10"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the pivot field by its name in the collection.
PivotField field = pivotTable.Fields["State"];
// Add calculated items to the "State" field.
field.CalculatedItems.Add("=Arizona+California+Colorado", "West Total");
field.CalculatedItems.Add("=Illinois+Kansas+Wisconsin", "Midwest Total");
```

Visual Basic

```
(PivotCalculatedItemActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report10")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the pivot field by its name in the collection.
Dim field As PivotField = pivotTable.Fields("State")
' Add calculated items to the "State" field.
field.CalculatedItems.Add("=Arizona+California+Colorado", "West Total")
field.CalculatedItems.Add("=Illinois+Kansas+Wisconsin", "Midwest Total")
```

The resulting PivotTable report is shown in the image below (the workbook is opened in Microsoft® Excel®).

Row Labels	Sum of Sales
Arizona	\$ 20,736.00
Dairy Products	\$ 11,997.00
Grains/Cereals	\$ 8,739.00
California	\$ 23,900.00
Dairy Products	\$ 13,195.00
Grains/Cereals	\$ 10,705.00
Colorado	\$ 23,685.00
Dairy Products	\$ 13,650.00
Grains/Cereals	\$ 10,035.00
Illinois	\$ 24,165.00
Dairy Products	\$ 13,720.00
Grains/Cereals	\$ 10,445.00
Kansas	\$ 22,230.00
Dairy Products	\$ 12,625.00
Grains/Cereals	\$ 9,605.00
Wisconsin	\$ 21,156.00
Dairy Products	\$ 11,045.00
Grains/Cereals	\$ 10,111.00
West Total	\$ 68,321.00
Dairy Products	\$ 38,842.00
Grains/Cereals	\$ 29,479.00
Midwest Total	\$ 67,551.00
Dairy Products	\$ 37,390.00
Grains/Cereals	\$ 30,161.00
Grand Total	\$ 271,744.00

Modify a Calculated Item

To change a formula for a calculated item, get access to the required item by its index in the PivotCalculatedItemCollection collection and then assign a new formula to the item's PivotItem.Formula property. To rename a calculated item, use the PivotItem.Caption property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#	
----	--

```
(PivotCalculatedItemActions.cs)
Worksheet worksheet = workbook.Worksheets["Report7"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the pivot field by its name in the collection.
PivotField field = pivotTable.Fields["Customer"];
// Add a calculated item to the "Customer" field.
PivotItem item = field.CalculatedItems.Add("='Big Foods'*110%", "Big Foods");
//Change the formula for the calculated item.
item.Formula = "='Big Foods'*115%";
```

Visual Basic

```
(PivotCalculatedItemActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report7")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the pivot field by its name in the collection.
Dim field As PivotField = pivotTable.Fields("Customer")
' Add a calculated item to the "Customer" field.
Dim item As PivotItem = field.CalculatedItems.Add("='Big Foods'*110%", "Bi")
'Change the formula for the calculated item.
item.Formula = "='Big Foods'*115%"
```

Remove a Calculated Item

To remove a calculated item from the field, use the `PivotCalculatedItemCollection.Remove` or `PivotCalculatedItemCollection.RemoveAt` method. To remove all calculated items from the collection at once, use the `PivotCalculatedItemCollection.Clear` method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotCalculatedItemActions.cs)
Worksheet worksheet = workbook.Worksheets["Report7"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the pivot field by its name in the collection.
PivotField field = pivotTable.Fields["Customer"];
// Add a calculated item to the "Customer" field.
field.CalculatedItems.Add("='Big Foods'*110%", "Big Foods Sales Plan");
//Remove the calculated item by its index from the collection.
field.CalculatedItems.RemoveAt(0);
```

Visual Basic

```
(PivotCalculatedItemActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report7")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the pivot field by its name in the collection.
Dim field As PivotField = pivotTable.Fields("Customer")
' Add a calculated item to the "Customer" field.
field.CalculatedItems.Add("='Big Foods'*110%", "Big Foods Sales Plan")
'Remove the calculated item by its index from the collection.
field.CalculatedItems.RemoveAt(0)
```

How to: Sort Items in a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Sort Items in a Pivot Table](#)

The examples below demonstrate how to sort items in PivotTable fields by [labels](#) and [summary values](#).

Sort Item Labels

To sort items in a specific PivotTable field in ascending or descending order, assign the corresponding PivotFieldSortType enumeration member to the PivotField.SortType property.

For example, the following code sorts items of the "Product" field in ascending order.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotFieldActions.cs)
Worksheet worksheet = workbook.Worksheets["Report1"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the pivot field by its name in the collection.
PivotField field = pivotTable.Fields["Product"];
// Sort items in the "Product" field.
field.SortType = PivotFieldSortType.Ascending;
```

Visual Basic

```
(PivotFieldActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report1")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the pivot field by its name in the collection.
Dim field As PivotField = pivotTable.Fields("Product")
' Sort items in the "Product" field.
field.SortType = PivotFieldSortType.Ascending
```

Sort Items by Summary Values

Besides sorting item labels, you can also sort items in a row or column field based on values in the [data area](#) of the PivotTable report. To do this, call the PivotField.SortItems method for the field you wish to sort and pass the following parameters.

- A PivotFieldSortType enumeration member that specifies the sort order (ascending or descending).
- A zero-based index of the data field in the PivotDataFieldCollection collection by which items in the field should be sorted.
- An array or collection of the PivotItemReference objects that represent references to specific PivotTable items describing a particular row or column in the data area containing values to sort by. If this parameter is not specified, items in the field are sorted by their grand total values.

Note

If you try to apply sorting to a field that is not currently shown as a row or column field in the report, or the sort settings you specify are invalid, an exception occurs. To view sort options applied to a specific field, use the field's PivotField.Sort property.

The following code example shows how to sort products in descending order by sales values in the first quarter.

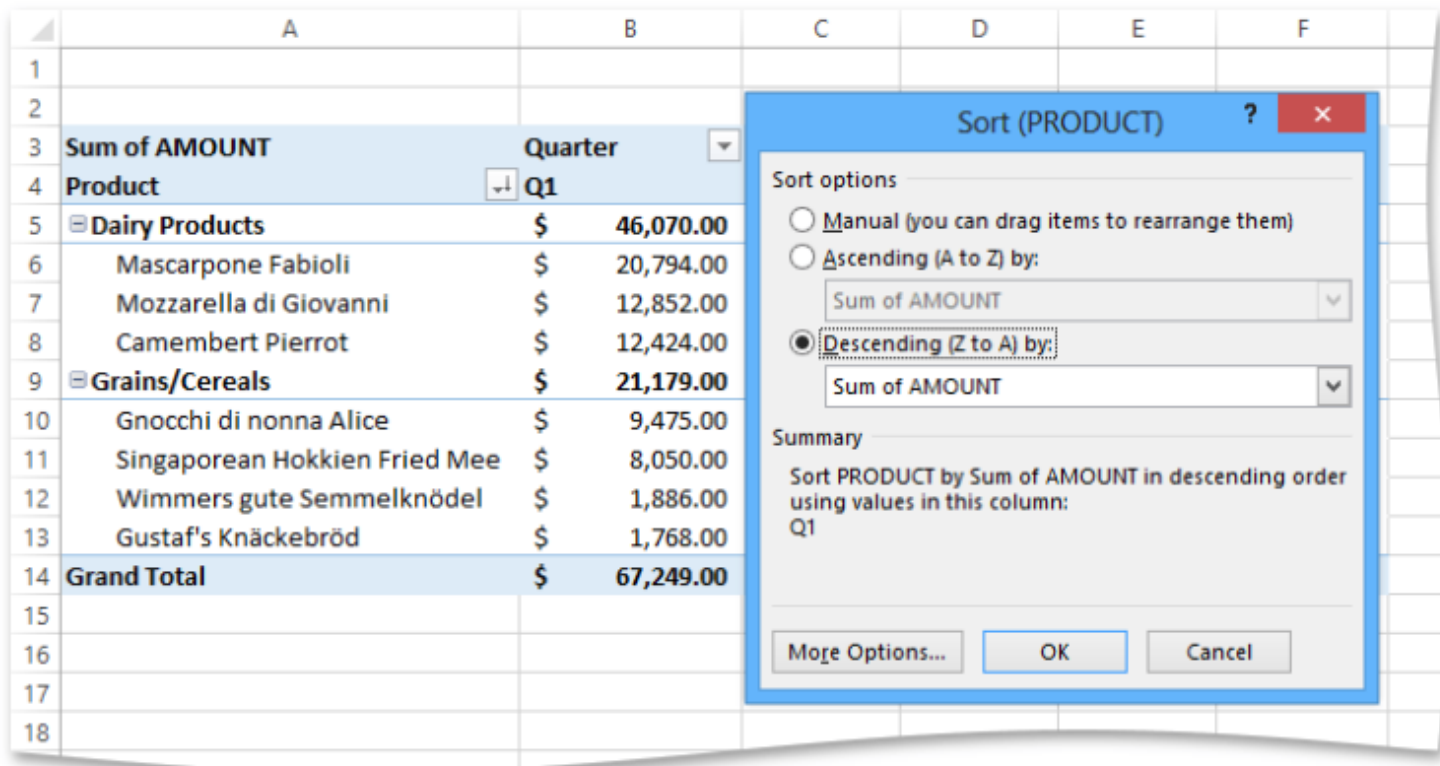
C#

```
// Access the pivot table by its index in the collection.
PivotTable pivotTable = worksheet.PivotTables[0];
// Create a reference to the "Q1" item of the "Quarter" field.
PivotItemReference item = new PivotItemReference(0, 0);
// Sort the "Product" field by the "Sum of Amount" field in descending order.
pivotTable.Fields["Product"].SortItems(PivotFieldSortType.Descending, 0, item);
```

Visual Basic

```
' Access the pivot table by its index in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables(0)
' Create a reference to the "Q1" item of the "Quarter" field.
Dim item As New PivotItemReference(0, 0)
' Sort the "Product" field by the "Sum of Amount" field in descending order.
pivotTable.Fields("Product").SortItems(PivotFieldSortType.Descending, 0, item);
```

The following image shows the result of the code's execution (the workbook is opened in Microsoft® Excel®).



How to: Filter Items in a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Filter Items in a Pivot Table](#)

This topic describes how to filter items in a PivotTable report. Different types of filters are available: you show or hide specific items, construct the filter expression to display item labels that meet the given criteria (**Label Filters** and **Date Filters**), or filter a field based on summary values in the [data area](#) (**Value Filters**).

The table below describes API members that allow you to apply a filter to a PivotTable field.

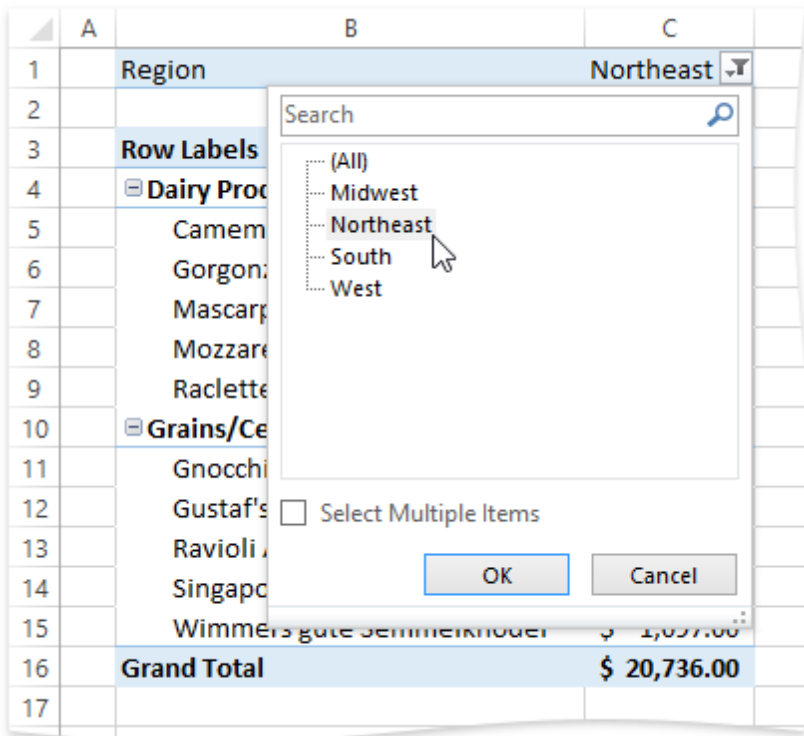
Member	Description
PivotField.ShowSingleItem	Displays the specified item in the PivotTable field.
PivotItem.Visible	Specifies whether the current item is displayed in the field.
PivotTable.Filters	Provides access to the collection of filters applied to a pivot table.
PivotFilterCollection.Add	Applies a filter to the specified row or column field.
PivotFilterType	Specifies the type of the filter to be applied to the PivotTable field.
PivotFilter	Represents a filter applied to the PivotTable field.
PivotFilter.Field	Returns the PivotTable field to which the current filter is applied.
PivotFilter.Value	Returns the first filter criteria value.
PivotFilter.SecondValue	Returns the second filter criteria value.
PivotFilter.MeasureField	Returns the data field used by the current value filter.
PivotFilter.Top10Type	Specifies the type of the "Top 10" value filter.

Select the task you wish to perform.

- [Show or Hide Specific Items](#)
- [Use a Label Filter](#)
- [Use a Date Filter](#)
- [Use a Value Filter](#)
- [Use a Top 10 Filter](#)
- [Use Multiple Filters per Field](#)
- [Remove a Filter](#)

Show or Hide Specific Items

The following example uses a report filter to filter the entire pivot table to show sales data for the Northeastern region. Before using the report filter, make sure that the corresponding field is added to the [report filter area](#) of your PivotTable report.



```
C#
Worksheet worksheet = workbook.Worksheets["Report4"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Move the "Region" field to the report filter area.
pivotTable.PageFields.Add(pivotTable.Fields["Region"]);
// Select the second (Northeast) item in the "Region" field.
pivotTable.Fields["Region"].ShowSingleItem(1);
```

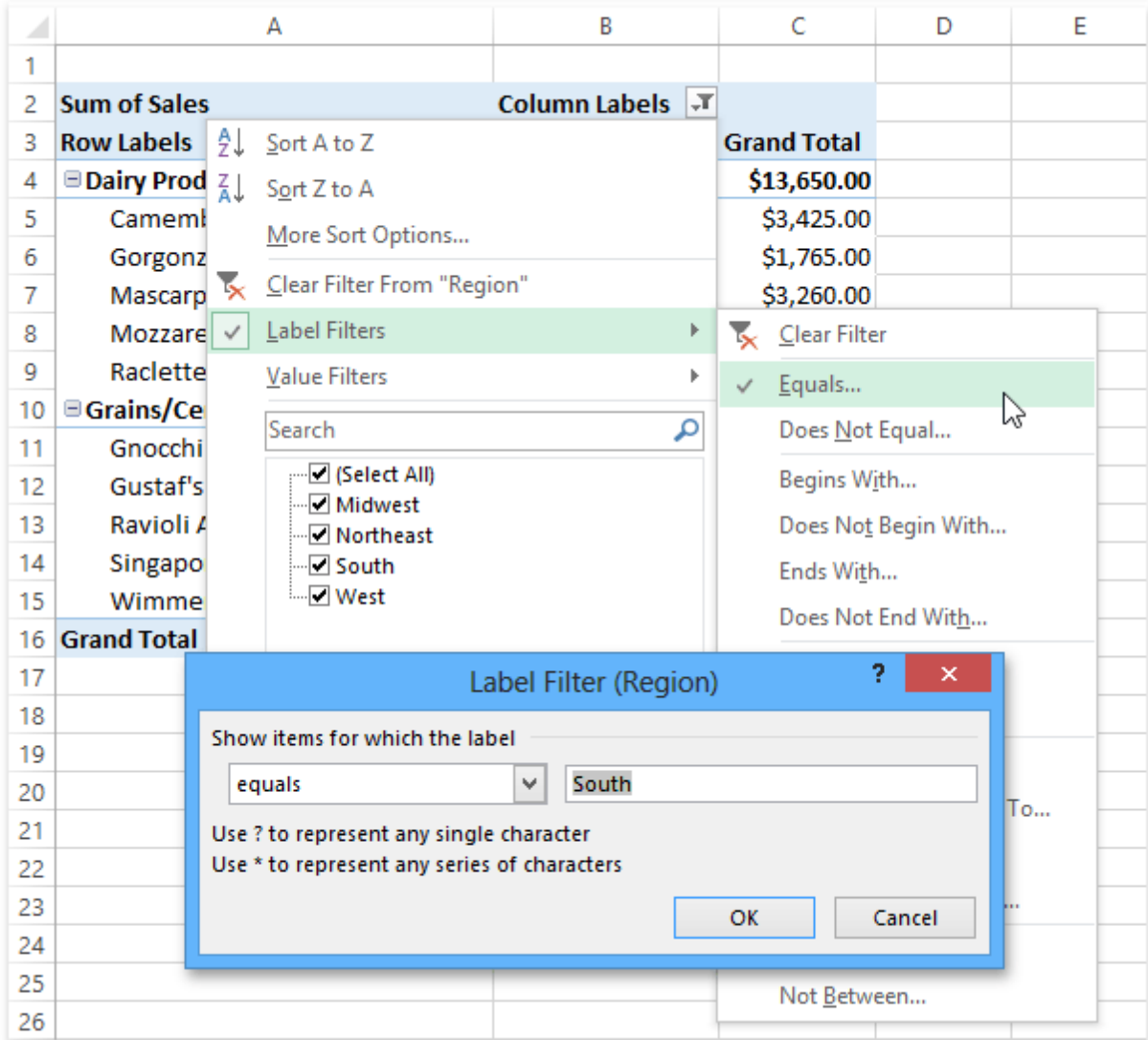
```
Visual Basic
Dim worksheet As Worksheet = workbook.Worksheets("Report4")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Move the "Region" field to the report filter area.
pivotTable.PageFields.Add(pivotTable.Fields("Region"))
' Select the second (Northeast) item in the "Region" field.
pivotTable.Fields("Region").ShowSingleItem(1)
```

Use a Label Filter

The following example demonstrates how to apply a label filter to the "Region" column field to display sales data only for the Southern region.

Tip

To perform more versatile filtering, you can use *wildcard* characters. The asterisk * matches any number of characters, while the question mark ? represents a single character. To filter labels containing a specific character, such as the asterisk, question mark or tilde, put the tilde (~) before it.



Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Report4"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the "Region" field.
PivotField field = pivotTable.Fields[0];
// Filter the "Region" field by text to display sales data for the "South" region.
pivotTable.Filters.Add(field, PivotFilterType.CaptionEqual, "South");
```

Visual Basic

```
(PivotTableFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report4")
worksheet.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the "Region" field.
Dim field As PivotField = pivotTable.Fields(0)
' Filter the "Region" field by text to display sales data for the "South" region.
pivotTable.Filters.Add(field, PivotFilterType.CaptionEqual, "South")
```

Use a Date Filter

The following example demonstrates how to apply a filter to the "Date" row field to show sales data for the second quarter.

	A	B	C	D	E
1					
2		Sum of AMOUNT	Column Labels		
3		Row Labels	Dairy Products	Grains/Cereals	Grand Total
4		Select field:	8270		8270
5		DATE	8270		8270
6		Sort Oldest to Newest	8270		8270
7		Sort Newest to Oldest	11100	3980	15080
8		More Sort Options...	11100		11100
9		Clear Filter From "DATE"	4850		4850
10			6250		6250
11		Date Filters		3980	3980
12		Value Filters		3980	3980
13					9166
14		Search			9166
15		(Select All)			9166
16		1/1/2014		2592	5914
17		1/7/2014			3322
18		1/11/2014			3322
19		1/25/2014		2592	2592
20		2/10/2014		2592	2592
21		3/8/2014		6989	6989
22		3/14/2014		6989	6989
23		3/24/2014		Quarter 1	44
24		4/9/2014		Quarter 2	45
25				Quarter 3	66
26					
27					
28					
29					
30					
31					
32					
33					
34					
35					
36					
37					
38					
39					

OK

Cancel

Clear Filter

Equals...

Before...

After...

Between...

Tomorrow

Today

Yesterday

Next Week

This Week

Last Week

Next Month

Year to Date

All Dates in the Period

Custom Filter...

August

September

October

November

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

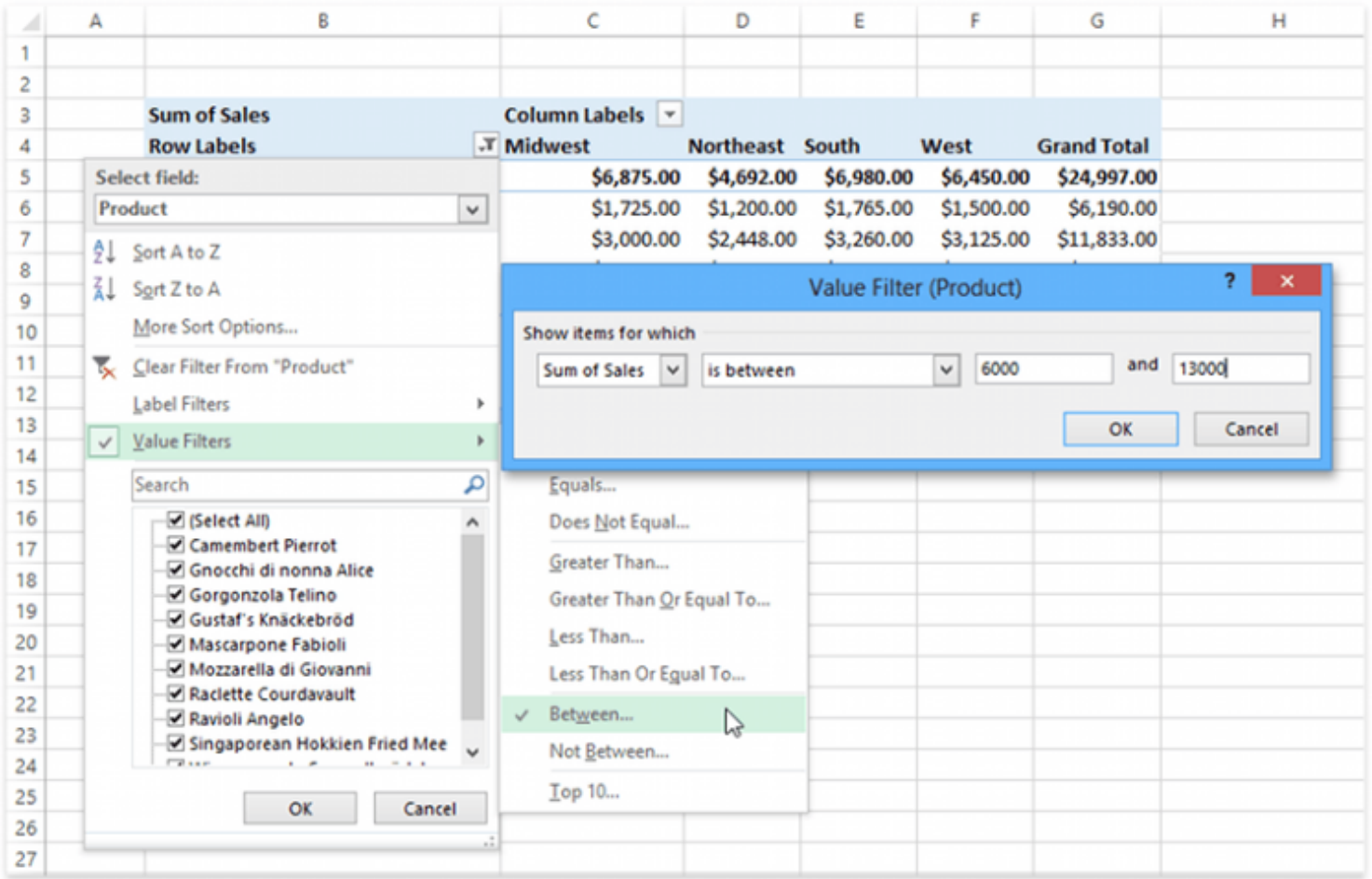
```
(PivotTableFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Report6"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the "Date" field.
PivotField field = pivotTable.Fields[0];
// Filter the "Date" field to display sales for the second quarter.
pivotTable.Filters.Add(field, PivotFilterType.SecondQuarter);
```

Visual Basic

```
(PivotTableFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report6")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the "Date" field.
Dim field As PivotField = pivotTable.Fields(0)
' Filter the "Date" field to display sales for the second quarter.
pivotTable.Filters.Add(field, PivotFilterType.SecondQuarter)
```

Use a Value Filter

The following example demonstrates how to apply a value filter to the "Product" row field to display products whose total sales fall between \$6000 and \$13000.



Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#	
<pre>(PivotTableFilterActions.cs) Worksheet worksheet = workbook.Worksheets["Report4"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Access the pivot table by its name in the collection. PivotTable pivotTable = worksheet.PivotTables["PivotTable1"]; // Access the "Product" field. PivotField field = pivotTable.Fields[1]; // Filter the "Product" field to display products with grand total sales b pivotTable.Filters.Add(field, pivotTable.DataFields[0], PivotFilterType.Va</pre>	
Visual Basic	

```
(PivotTableFilterActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report4")
worksheet.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the "Product" field.
Dim field As PivotField = pivotTable.Fields(1)
' Filter the "Product" field to display products with grand total sales be
pivotTable.Filters.Add(field, pivotTable.DataFields(0), PivotFilterType.Va
```

Use a Top 10 Filter

A "Top 10" value filter allows you to display the specified number of top or bottom items in a field (PivotFilterType.Count), show the top or bottom values that contribute to the specified percent of the filtered field's grand total (PivotFilterType.Percent), or filter the top or bottom values that make up a specific sum (PivotFilterType.Sum).

The following example demonstrates how to apply the "Top 10" filter to the "Product" row field to display two products with the lowest total sales.

Select field:

Product

Sort A to Z

Sort Z to A

More Sort Options...

Clear Filter From "Product"

Label Filters

Value Filters

Search

(Select All)

Camembert Pierrot

Gnocchi di nonna Alice

Gorgonzola Telino

Gustaf's Knäckebröd

Mascarpone Fabioli

Mozzarella di Giovanni

Raclette Courdavault

Ravioli Angelo

Singaporean Hokkien Fried Mee

OK

Cancel

Top 10 Filter (Product)

Show

Bottom

2

Items

by

Sum of Sales

OK

Cancel

Greater Than...

Greater Than Or Equal To...

Less Than...

Less Than Or Equal To...

Between...

Not Between...

Top 10...

	A	B	C	D	E	F	G
1							
2		Sum of Sales	Column Labels				
3		Row Labels	Midwest	Northeast	South	West	Grand Total
4			\$2,915.00	\$2,313.00	\$2,890.00	\$2,975.00	\$11,093.00
5			\$1,650.00	\$1,216.00	\$1,435.00	\$1,675.00	\$5,976.00
6			\$1,265.00	\$1,097.00	\$1,455.00	\$1,300.00	\$5,117.00
7			\$2,915.00	\$2,313.00	\$2,890.00	\$2,975.00	\$11,093.00
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							

Show Me

A complete sample project is available in the DevExpress Code Examples database at

© 2018 Developer Express Inc.

437

<http://www.devexpress.com/example=T501894>.

C#
<pre>(PivotTableFilterActions.cs) Worksheet worksheet = workbook.Worksheets["Report4"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Access the pivot table by its name in the collection. PivotTable pivotTable = worksheet.PivotTables["PivotTable1"]; // Access the "Product" field. PivotField field = pivotTable.Fields[1]; // Filter the "Product" field to display two products with the lowest sales PivotFilter filter = pivotTable.Filters.Add(field, pivotTable.DataFields[0] filter.Top10Type = PivotFilterTop10Type.Bottom;</pre>

Visual Basic
<pre>(PivotTableFilterActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("Report4") workbook.Worksheets.ActiveWorksheet = worksheet ' Access the pivot table by its name in the collection. Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1") ' Access the "Product" field. Dim field As PivotField = pivotTable.Fields(1) ' Filter the "Product" field to display two products with the lowest sales Dim filter As PivotFilter = pivotTable.Filters.Add(field, pivotTable.DataF filter.Top10Type = PivotFilterTop10Type.Bottom</pre>

Use Multiple Filters per Field

To enable the capability to apply multiple filters to a single row or column field, set the `PivotBehaviorOptions.AllowMultipleFieldFilters` property to **true**. By default, this property is **false** and if you try to apply more than one filter to a PivotTable field, only the last specified filter will be applied.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotTableFilterActions.cs)
Worksheet worksheet = workbook.Worksheets["Report6"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Allow multiple filters for a field.
pivotTable.Behavior.AllowMultipleFieldFilters = true;
// Filter the "Date" field to display sales for the second quarter.
PivotField field1 = pivotTable.Fields[0];
pivotTable.Filters.Add(field1, PivotFilterType.SecondQuarter);
// Add the second filter to the "Date" field to display two days with the
PivotFilter filter = pivotTable.Filters.Add(field1, pivotTable.DataFields[
filter.Top10Type = PivotFilterTop10Type.Bottom;
```

Visual Basic	
<pre>(PivotTableFilterActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("Report6") workbook.Worksheets.ActiveWorksheet = worksheet ' Access the pivot table by its name in the collection. Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1") ' Allow multiple filters for a field. pivotTable.Behavior.AllowMultipleFieldFilters = True ' Filter the "Date" field to display sales for the second quarter. Dim field1 As PivotField = pivotTable.Fields(0) pivotTable.Filters.Add(field1, PivotFilterType.SecondQuarter) ' Add the second filter to the "Date" field to display two days with the 1 Dim filter As PivotFilter = pivotTable.Filters.Add(field1, pivotTable.Data filter.Top10Type = PivotFilterTop10Type.Bottom</pre>	

Remove a Filter

• Clear a specific filter

You can remove a filter applied to a PivotTable field using one of the methods listed in the table below.

Method	Description
PivotFilter.Delete	Removes the current filter from the collection of filters applied to a pivot table.
PivotFilterCollection.Remove	Removes the specified filter from the collection of filters applied to a pivot table.
PivotFilterCollection.RemoveAt	Removes the filter at the specified index from the collection of filters applied to a pivot table.
PivotField.ShowAllItems	Redisplays all items in the specified PivotTable field.

C#	
----	--

```
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Apply a filter to the "Region" field.
PivotFilter labelFilter = pivotTable.Filters.Add(pivotTable.Fields["Region"]);
// Remove the specified filter.
labelFilter.Delete();
```

Visual Basic

```
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Apply a filter to the "Region" field.
Dim labelFilter As PivotFilter = pivotTable.Filters.Add(pivotTable.Fields("Region"))
' Remove the specified filter.
labelFilter.Delete()
```

• Clear all filters

To remove all filters applied to the row and column fields of the PivotTable report at once, use the PivotFilterCollection.Clear method.

C#

```
// Clear all filters applied to the PivotTable fields.
pivotTable.Filters.Clear();
```

Visual Basic

```
' Clear all filters applied to the PivotTable fields.
pivotTable.Filters.Clear()
```

How to: Group Items in a Pivot Table

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pivot Tables](#) > [How to: Group Items in a Pivot Table](#)

Grouping can help you to get a clearer view of data and show only data you want to analyze. Use the `PivotField.GroupItems` method overloads to group data in a `PivotTable` report.

Select the task you wish to perform.

- [Group a Pivot Table by Date](#)
- [Group a Pivot Table by Numbers](#)
- [Group Selected Items](#)
- [Ungroup Data](#)

Group a Pivot Table by Date

The following example demonstrates how to group items in the "Date" field by quarters and months. To do this, use the `PivotField.GroupItems` method with the combination of `PivotFieldGroupByType.Quarters` and `PivotFieldGroupByType.Months` values passed as a parameter.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```
(PivotFieldGroupingActions.cs)
Worksheet worksheet = workbook.Worksheets["Report8"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection.
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the "DATE" field by its name in the collection.
PivotField field = pivotTable.Fields["DATE"];
// Group field items by quarters and months.
field.GroupItems(PivotFieldGroupByType.Quarters | PivotFieldGroupByType.Months);
```

Visual Basic

```
(PivotFieldGroupingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report8")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the "DATE" field by its name in the collection.
Dim field As PivotField = pivotTable.Fields("DATE")
' Group field items by quarters and months.
field.GroupItems(PivotFieldGroupByType.Quarters Or PivotFieldGroupByType.Months)
```

The image below shows the resulting report (the workbook is opened in Microsoft® Excel®).

Sum of AMOUNT		Column Labels				
Row Labels		A&B Supermarkets	Big Foods	Food Land	Miller's	Grand Total
Qtr1						
Jan	\$	8,330.00		\$ 11,242.00	\$ 8,002.00	\$ 27,574.00
Feb	\$	1,768.00				\$ 1,768.00
Mar	\$	5,750.00	\$ 12,602.00	\$ 8,475.00	\$ 11,080.00	\$ 37,907.00
Qtr2						
Apr	\$	8,270.00				\$ 8,270.00
May	\$	11,100.00		\$ 9,166.00	\$ 3,980.00	\$ 24,246.00
Jun	\$	10,311.00	\$ 2,592.00			\$ 12,903.00
Grand Total	\$	45,529.00	\$ 15,194.00	\$ 28,883.00	\$ 23,062.00	\$ 112,668.00

Group a Pivot Table by Numbers

To group a numeric field in code, use the `PivotField.GroupItems` method with the `PivotFieldGroupByType.NumericRanges` parameter, specify the smallest and largest number to group the field and an interval for each group.

The following example groups the sales amounts by thousands.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#	
<pre>(PivotFieldGroupingActions.cs) Worksheet worksheet = workbook.Worksheets["Report12"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Access the pivot table by its name in the collection. PivotTable pivotTable = worksheet.PivotTables["PivotTable1"]; // Access the "Usual Hours Worked" field by its name in the collection. PivotField field = pivotTable.Fields["Sales"]; // Group field items from 1000 to 4000 by 1000. field.GroupItems(1000, 4000, 1000, PivotFieldGroupByType.NumericRanges);</pre>	
Visual Basic	
<pre>(PivotFieldGroupingActions.vb) Dim worksheet As Worksheet = workbook.Worksheets("Report12") workbook.Worksheets.ActiveWorksheet = worksheet ' Access the pivot table by its name in the collection. Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1") ' Access the "Usual Hours Worked" field by its name in the collection. Dim field As PivotField = pivotTable.Fields("Sales") ' Group field items from 1000 to 4000 by 1000. field.GroupItems(1000, 4000, 1000, PivotFieldGroupByType.NumericRanges)</pre>	

The image below shows the resulting report (the workbook is opened in Microsoft® Excel®).

Count of Sales		Column Labels		
Row Labels	1000-1999	2000-2999	3000-3999	Grand Total
Camembert Pierrot		3	3	6
Gnocchi di nonna Alice				6
Gorgonzola Telino				6
Gustaf's Knäckebröd				6
Mascarpone Fabioli				6
Mozzarella di Giovanni				6
Raclette Courdavault				6
Ravioli Angelo				6
Singaporean Hokkien Fried Mee				6
Wimmers gute Semmelknödel				6
Grand Total	29	18	13	60

Group Selected Items

The following example demonstrates how to create a group of the first three items in the "State" field (Arizona, California and Colorado). To do this, use the PivotField.GroupItems method with a list of item indices (0, 1, 2) passed as a parameter.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#	
<pre>(PivotFieldGroupingActions.cs) Worksheet worksheet = workbook.Worksheets["Report11"]; workbook.Worksheets.ActiveWorksheet = worksheet; // Access the pivot table by its name in the collection. PivotTable pivotTable = worksheet.PivotTables["PivotTable1"]; // Access the "State" field by its name in the collection. PivotField field = pivotTable.Fields["State"]; // Add the "State" field to the column axis area. pivotTable.ColumnFields.Add(field); // Group the first three items in the field. IEnumerable<int> items = new List<int>() { 0, 1, 2 }; field.GroupItems(items); // Access the created grouped field by its index in the field collection. int groupedFieldIndex = pivotTable.Fields.Count - 1; PivotField groupedField = pivotTable.Fields[groupedFieldIndex]; // Set the grouped item caption to "West". groupedField.Items[0].Caption = "West";</pre>	
Visual Basic	

```

(PivotFieldGroupingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report11")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection.
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the "State" field by its name in the collection.
Dim field As PivotField = pivotTable.Fields("State")
' Add the "State" field to the column axis area.
pivotTable.ColumnFields.Add(field)
' Group the first three items in the field.
Dim items As IEnumerable(Of Integer) = New List(Of Integer)() From {0, 1,
field.GroupItems(items)
' Access the created grouped field by its index in the field collection.
Dim groupedFieldIndex As Integer = pivotTable.Fields.Count - 1
Dim groupedField As PivotField = pivotTable.Fields(groupedFieldIndex)
' Set the grouped item caption to "West".
groupedField.Items(0).Caption = "West"

```

The image below shows the resulting report (the workbook is opened in Microsoft® Excel®).

Sum of Sales		Column Labels						
		West			Illinois	Kansas	Wisconsin	Grand Total
Row Labels		Arizona	California	Colorado	Illinois	Kansas	Wisconsin	
Dairy Products		\$ 11,997.00	\$ 13,195.00	\$ 13,650.00	\$ 13,720.00	\$ 12,625.00	\$ 11,045.00	\$ 76,232.00
Camembert Pierrot		\$ 3,425.00	\$ 2,820.00	\$ 3,425.00	\$ 3,520.00	\$ 2,550.00	\$ 2,900.00	\$ 18,640.00
Gorgonzola Telino		\$ 1,200.00	\$ 1,725.00	\$ 1,765.00	\$ 1,500.00	\$ 1,880.00	\$ 1,350.00	\$ 9,420.00
Mascarpone Fabioli		\$ 2,448.00	\$ 3,000.00	\$ 3,260.00	\$ 3,125.00	\$ 3,520.00	\$ 2,350.00	\$ 17,703.00
Mozzarella di Giovanni		\$ 1,044.00	\$ 2,150.00	\$ 1,955.00	\$ 1,825.00	\$ 1,725.00	\$ 1,265.00	\$ 9,964.00
Raclette Courdavault		\$ 3,880.00	\$ 3,500.00	\$ 3,245.00	\$ 3,750.00	\$ 2,950.00	\$ 3,180.00	\$ 20,505.00
Grains/Cereals		\$ 8,739.00	\$ 10,705.00	\$ 10,035.00	\$ 10,445.00	\$ 9,605.00	\$ 10,111.00	\$ 59,640.00
Gnocchi di nonna Alice		\$ 1,216.00	\$ 1,650.00	\$ 1,435.00	\$ 1,675.00	\$ 1,765.00	\$ 1,715.00	\$ 9,456.00
Gustaf's Knäckebröd		\$ 2,420.00	\$ 2,980.00	\$ 2,345.00	\$ 2,720.00	\$ 2,200.00	\$ 1,968.00	\$ 14,633.00
Ravioli Angelo		\$ 2,390.00	\$ 2,985.00	\$ 2,965.00	\$ 3,050.00	\$ 1,950.00	\$ 2,785.00	\$ 16,125.00
Singaporean Hokkien Fried Mee		\$ 1,616.00	\$ 1,825.00	\$ 1,835.00	\$ 1,700.00	\$ 1,635.00	\$ 1,546.00	\$ 10,157.00
Wimmers gute Semmelknödel		\$ 1,097.00	\$ 1,265.00	\$ 1,455.00	\$ 1,300.00	\$ 2,055.00	\$ 2,097.00	\$ 9,269.00
Grand Total		\$ 20,736.00	\$ 23,900.00	\$ 23,685.00	\$ 24,165.00	\$ 22,230.00	\$ 21,156.00	\$ 135,872.00

Ungroup Data

To ungroup data in a pivot table, use the `PivotField.UngroupItems` method overloads.

The following example creates two groups of "State" items and subsequently ungroups the first group.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T501894>.

C#

```

(PivotFieldGroupingActions.cs)
Worksheet worksheet = workbook.Worksheets["Report11"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Access the pivot table by its name in the collection
PivotTable pivotTable = worksheet.PivotTables["PivotTable1"];
// Access the "State" field by its name in the collection.
PivotField field = pivotTable.Fields["State"];
// Add the "State" field to the column axis area.
pivotTable.ColumnFields.Add(field);
// Group the first three items in the field.
IEnumerable<int> items = new List<int>() { 0, 1, 2 };
field.GroupItems(items);
// Access the created grouped field by its index in the field collection.
int groupedFieldIndex = pivotTable.Fields.Count - 1;
PivotField groupedField = pivotTable.Fields[groupedFieldIndex];
// Set the grouped item caption to "West".
groupedField.Items[0].Caption = "West";
// Group the remaining field items.
items = new List<int>() { 3, 4, 5 };
field.GroupItems(items);
// Set the grouped item caption to "Midwest"
groupedField.Items[1].Caption = "Midwest";
// Ungroup the "West" item.
items = new List<int> { 0 };
groupedField.UngroupItems(items);

```

Visual Basic

```

(PivotFieldGroupingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("Report11")
workbook.Worksheets.ActiveWorksheet = worksheet
' Access the pivot table by its name in the collection
Dim pivotTable As PivotTable = worksheet.PivotTables("PivotTable1")
' Access the "State" field by its name in the collection.
Dim field As PivotField = pivotTable.Fields("State")
' Add the "State" field to the column axis area.
pivotTable.ColumnFields.Add(field)
' Group the first three items in the field.
Dim items As IEnumerable(Of Integer) = New List(Of Integer)() From {0, 1, 2}
field.GroupItems(items)
' Access the created grouped field by its index in the field collection.
Dim groupedFieldIndex As Integer = pivotTable.Fields.Count - 1
Dim groupedField As PivotField = pivotTable.Fields(groupedFieldIndex)
' Set the grouped item caption to "West".
groupedField.Items(0).Caption = "West"
' Group the remaining field items.
items = New List(Of Integer)() From {3, 4, 5}
field.GroupItems(items)
' Set the grouped item caption to "Midwest"
groupedField.Items(1).Caption = "Midwest"
' Ungroup the "West" item.
items = New List(Of Integer) From {0}
groupedField.UngroupItems(items)

```

The image below shows the resulting report (the workbook is opened in Microsoft® Excel®).

Sum of Sales	Column Labels						Grand Total	
	Arizona	California	Colorado	Midwest				
Row Labels	Arizona	California	Colorado	Illinois	Kansas	Wisconsin		
Dairy Products	\$ 11,997.00	\$ 13,195.00	\$ 13,650.00	\$ 13,720.00	\$ 12,625.00	\$ 11,045.00	\$	76,232.00
Camembert Pierrot	\$ 3,425.00	\$ 2,820.00	\$ 3,425.00	\$ 3,520.00	\$ 2,550.00	\$ 2,900.00	\$	18,640.00
Gorgonzola Telino	\$ 1,200.00	\$ 1,725.00	\$ 1,765.00	\$ 1,500.00	\$ 1,880.00	\$ 1,350.00	\$	9,420.00
Mascarpone Fabioli	\$ 2,448.00	\$ 3,000.00	\$ 3,260.00	\$ 3,125.00	\$ 3,520.00	\$ 2,350.00	\$	17,703.00
Mozzarella di Giovanni	\$ 1,044.00	\$ 2,150.00	\$ 1,955.00	\$ 1,825.00	\$ 1,725.00	\$ 1,265.00	\$	9,964.00
Raclette Courdavault	\$ 3,880.00	\$ 3,500.00	\$ 3,245.00	\$ 3,750.00	\$ 2,950.00	\$ 3,180.00	\$	20,505.00
Grains/Cereals	\$ 8,739.00	\$ 10,705.00	\$ 10,035.00	\$ 10,445.00	\$ 9,605.00	\$ 10,111.00	\$	59,640.00
Gnocchi di nonna Alice	\$ 1,216.00	\$ 1,650.00	\$ 1,435.00	\$ 1,675.00	\$ 1,765.00	\$ 1,715.00	\$	9,456.00
Gustaf's Knäckebröd	\$ 2,420.00	\$ 2,980.00	\$ 2,345.00	\$ 2,720.00	\$ 2,200.00	\$ 1,968.00	\$	14,633.00
Ravioli Angelo	\$ 2,390.00	\$ 2,985.00	\$ 2,965.00	\$ 3,050.00	\$ 1,950.00	\$ 2,785.00	\$	16,125.00
Singaporean Hokkien Fried Mee	\$ 1,616.00	\$ 1,825.00	\$ 1,835.00	\$ 1,700.00	\$ 1,635.00	\$ 1,546.00	\$	10,157.00
Wimmers gute Semmelknödel	\$ 1,097.00	\$ 1,265.00	\$ 1,455.00	\$ 1,300.00	\$ 2,055.00	\$ 2,097.00	\$	9,269.00
Grand Total	\$ 20,736.00	\$ 23,900.00	\$ 23,685.00	\$ 24,165.00	\$ 22,230.00	\$ 21,156.00	\$	135,872.00

Printing

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Printing](#)

This section contains the following examples:

- [How to: Print a Workbook](#)
- [How to: Specify Print Settings](#)
- [How to: Show a Print Preview Form for a Workbook](#)
- [How to: Add Headers and Footers to a Worksheet Printout](#)
- [How to: Define a Print Area](#)
- [How to: Print Titles on a Worksheet](#)
- [How to: Use the WPF Chart Rendering Mechanism When Printing or Exporting a Workbook to PDF](#)

How to: Print a Workbook

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Printing](#) > [How to: Print a Workbook](#)

The table below lists methods that allow you to print a [Workbook](#) or individual worksheets.

Method	Description
Workbook.Print	Prints a document using the default or custom printer settings defined by the System.Drawing.Printing.PrinterSettings class instance.
Sheet.Print	Prints a particular sheet in the document.

C#

```
using DevExpress.Spreadsheet;
using System.Drawing.Printing;
// ...
// Create a new Workbook object.
Workbook workbook = new Workbook();
// Load a document from a file.
workbook.LoadDocument("Documents\\Document.xlsx");
// Create an object containing printer settings.
PrinterSettings printerSettings = new PrinterSettings();
// Define the printer to use.
printerSettings.PrinterName = "Microsoft Print to PDF";
printerSettings.PrintToFile = true;
printerSettings.PrintFileName = "Documents\\PrintedDocument.pdf";
// Specify that the first three pages should be printed.
printerSettings.PrintRange = PrintRange.SomePages;
printerSettings.FromPage = 1;
printerSettings.ToPage = 3;
// Set the number of copies to print.
printerSettings.Copies = 1;
// Print the workbook using the specified printer settings.
workbook.Print(printerSettings);
```

Visual Basic

```
Imports DevExpress.Spreadsheet
Imports System.Drawing.Printing
' ...
' Create a new Workbook object.
Private workbook As New Workbook()
' Load a document from a file.
workbook.LoadDocument("Documents\\Document.xlsx")
' Create an object containing printer settings.
Dim printerSettings As New PrinterSettings()
' Define the printer to use.
printerSettings.PrinterName = "Microsoft Print to PDF"
printerSettings.PrintToFile = True
printerSettings.PrintFileName = "Documents\\PrintedDocument.pdf"
' Specify that the first three pages should be printed.
printerSettings.PrintRange = PrintRange.SomePages
printerSettings.FromPage = 1
printerSettings.ToPage = 3
' Set the number of copies to print.
printerSettings.Copies = 1
' Print the workbook using the specified printer settings.
workbook.Print(printerSettings)
```

You can also specify various printout options to control how the document is printed. To do this, use properties of the WorksheetView and WorksheetPrintOptions objects as described in the [How to: Specify Print Settings](#) example.

See Also

[How to: Specify Print Settings](#)

[How to: Add Headers and Footers to a Worksheet Printout](#)

[How to: Define a Print Area](#)

[How to: Print Titles on a Worksheet](#)

How to: Specify Print Settings

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Printing](#) > [How to: Specify Print Settings](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to set print options to create the required printout.

Use the `WorksheetView` properties to set general page options. The active view for a worksheet is accessible using the `Worksheet.ActiveView` property. Worksheet View enables you to set orientation, margins and paper size settings, among others. View settings are used when a worksheet is being printed.

Worksheet printing options is a set of more print-specific options returned by the `Worksheet.PrintOptions` property.

To scale the content to fit in pages, set the `WorksheetPrintOptions.FitToPage` property to **true** and specify the desired number of pages by width (`WorksheetPrintOptions.FitToWidth` property) or by height (the `WorksheetPrintOptions.FitToHeight` property). You can scale the content arbitrarily by specifying the scale percentage using the `WorksheetPrintOptions.Scale` property, the `WorksheetPrintOptions.FitToPage` property is **false** in this case.

Other options include printing gridlines (the `WorksheetPrintOptions.PrintGridlines` property), printing in color or black-and-white (the `WorksheetPrintOptions.BlackAndWhite` property), and more.

Note

The `WorksheetPrintOptions.Draft`, `WorksheetPrintOptions.BlackAndWhite`, `WorksheetPrintOptions.NumberOfCopies` and `WorksheetPrintOptions.CommentsPrintMode` options are not supported when printing a document from the **Spreadsheet Document API**. However, they are saved to a file, so you can use Microsoft® Excel® or another spreadsheet application to load and print a document.

C#

(PrintingActions.cs)
worksheet.ActiveView.Orientation = PageOrientation.Landscape;
// Display row and column headings.
worksheet.ActiveView.ShowHeadings = true;
worksheet.ActiveView.PaperKind = System.Drawing.Printing.PaperKind.A4;
// Access an object providing print options.
WorksheetPrintOptions printOptions = worksheet.PrintOptions;
// Print in black and white.
printOptions.BlackAndWhite = true;
// Do not print gridlines.
printOptions.PrintGridlines = false;
// Scale the print area to fit to a page.
printOptions.FitToPage = true;
// Print a dash instead of a cell error message.
printOptions.ErrorsPrintMode = ErrorsPrintMode.Dash;

Visual Basic

```
(PrintingActions.vb)
worksheet.ActiveView.Orientation = PageOrientation.Landscape
' Display row and column headings.
worksheet.ActiveView.ShowHeadings = True
worksheet.ActiveView.PaperKind = System.Drawing.Printing.PaperKind.A4
' Access an object providing print options.
Dim printOptions As WorksheetPrintOptions = worksheet.PrintOptions
' Print in black and white.
printOptions.BlackAndWhite = True
' Do not print gridlines.
printOptions.PrintGridlines = False
' Scale the print area to fit to a page.
printOptions.FitToPage = True
' Print a dash instead of a cell error message.
printOptions.ErrorsPrintMode = ErrorsPrintMode.Dash
```

How to: Show a Print Preview Form for a Workbook

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Printing](#) > [How to: Show a Print Preview Form for a Workbook](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to use the DevExpress Printing Library to show a Print Preview for a document loaded into a [Workbook](#) instance (*workbook*, in this example). To do this, follow the steps below:

- Add a reference to the **DevExpress.XtraPrinting.v18.1.dll** assembly and explicitly import the **DevExpress.XtraPrinting** namespace into the code with a **using** directive (**Imports** in Visual Basic).
- Create a new `PrintingSystem` instance for creating and printing a document.
- Create a `PrintableComponentLink` printing link with the specified printing system and assign the workbook to the `PrintableComponentLinkBase.Component` property.
- Generate a document to print by calling the `LinkBase.CreateDocument` method.
- Call the `PrintableComponentLink.ShowPreviewDialog` method to invoke the print preview form that enables a user to print the document or save it as a PDF or graphic file.

C#

```
(PrintingActions.cs)
using DevExpress.Spreadsheet;
using DevExpress.XtraPrinting;

// Invoke the Print Preview dialog for the workbook.
using (PrintingSystem printingSystem = new PrintingSystem()) {
    using (PrintableComponentLink link = new PrintableComponentLink(printingSystem)) {
        link.Component = workbook;
        link.CreateDocument();
        link.ShowPreviewDialog();
    }
}
```

Visual Basic

```
(PrintingActions.vb)
Imports DevExpress.Spreadsheet
Imports DevExpress.XtraPrinting

' Invoke the Print Preview dialog for the workbook.
Using printingSystem As New PrintingSystem()
    Using link As New PrintableComponentLink(printingSystem)
        link.Component = workbook
        link.CreateDocument()
        link.ShowPreviewDialog()
    End Using
End Using
```

See Also
[How to: Print a Workbook](#)

How to: Add Headers and Footers to a Worksheet Printout

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Printing](#) > [How to: Add Headers and Footers to a Worksheet Printout](#)

The example below demonstrates how to insert headers and footers at the top and bottom of a worksheet printout.

To define the header and footer options, use the Worksheet.HeaderFooterOptions property. It provides access to the WorksheetHeaderFooterOptions object, which allows you to specify the header or footer for the first page (WorksheetHeaderFooterOptions.FirstHeader and WorksheetHeaderFooterOptions.FirstFooter), odd-numbered pages (WorksheetHeaderFooterOptions.OddHeader and WorksheetHeaderFooterOptions.OddFooter) and even-numbered pages (WorksheetHeaderFooterOptions.EvenHeader and WorksheetHeaderFooterOptions.EvenFooter) of the printed worksheet.

Header and Footer Options

Use the properties of the WorksheetHeaderFooterOptions object to specify the following header/footer settings.

- WorksheetHeaderFooterOptions.DifferentOddEven

Set this property to **true** to specify different headers and footers for the odd-numbered and even-numbered pages. If this property is **false**, all pages in a worksheet have the same headers or footers as specified by the WorksheetHeaderFooterOptions.OddHeader and WorksheetHeaderFooterOptions.OddFooter properties.

- WorksheetHeaderFooterOptions.DifferentFirst

Set this property to **true** to specify a unique header or footer for the first page of a worksheet.

- WorksheetHeaderFooterOptions.ScaleWithDoc

Set this property to **true** to scale headers and footers proportionately when you scale a worksheet to fit your information on the specified number of pages.

- WorksheetHeaderFooterOptions.AlignWithMargins

Set this property to **true** to align the header and footer edges with page margins

Header and Footer Codes

The SpreadsheetControl supports specific codes that enable including dynamic information into a header or footer, such as a page number, current date and time, filename, worksheet name, and so on. Note that these codes can also be obtained as constant fields or methods of the HeaderFooterCode class.

The supported header/footer codes are listed in the table below.

Code	HeaderFooterCode member	Description
&L	HeaderFooterCode.LeftSection	Left aligns the characters that follow.
&C	HeaderFooterCode.CenterSection	Centers the characters that follow.
&R	HeaderFooterCode.RightSection	Right aligns the characters that follow.
&P	HeaderFooterCode.PageNumber	Inserts the current page number.
&N	HeaderFooterCode.PageTotal	Inserts the total number of pages in a workbook.
&D	HeaderFooterCode.Date	Inserts the current date.
&T	HeaderFooterCode.Time	Inserts the current time.
&Z	HeaderFooterCode.WorkbookFilePath	Inserts the workbook file path.
&F	HeaderFooterCode.WorkbookFileName	Inserts the name of a workbook file.
&A	HeaderFooterCode.WorksheetName	Inserts the name of a worksheet.
&G	HeaderFooterCode.Picture	Inserts an image.
&&	HeaderFooterCode.Ampersand	Inserts the ampersand character.

&B	HeaderFooterCode.Bold	Turns bold on and off for the characters that follow.
&I	HeaderFooterCode.Italic	Turns italic on and off for the characters that follow.
&U	HeaderFooterCode.Underline	Turns underline on or off for the characters that follow.
&E	HeaderFooterCode.DoubleUnderline	Turns double underline on and off for the characters that follow.
&S	HeaderFooterCode.Strikethrough	Turns strikethrough on or off for the characters that follow.
&Y	HeaderFooterCode.Subscript	Turns subscript on or off for the characters that follow.
&X	HeaderFooterCode.Superscript	Turns superscript on or off for the characters that follow.
&"font name,font type"	HeaderFooterCode.Font	Prints the characters that follow using the specified font.
&font size	HeaderFooterCode.FontSize	Prints the characters that follow using the specified font size in points.
&KRRGGBB	HeaderFooterCode.FontColor	Prints the characters that follow using the specified color. Use the hexadecimal numbers to specify the color components.
&KTTSNN	HeaderFooterCode.FontColor	Prints the characters that follow using the specified color from the current theme. <i>TT</i> is the theme color Id, <i>S</i> is either "+" or "-" of the tint/shade value, and <i>NN</i> is the tint/shade value.

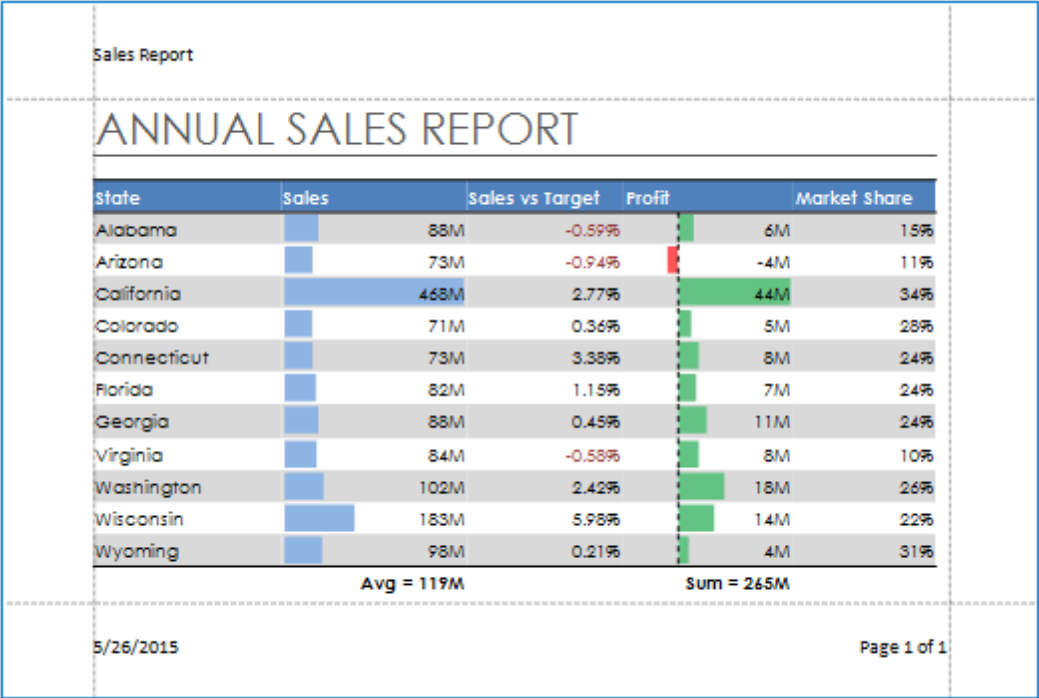
Example

The following example prints the workbook file name in the first page header, and inserts the current date and the page indicator in the first page footer.

C#	
<pre>WorksheetHeaderFooterOptions options = worksheet.HeaderFooterOptions; // Specify that the first page of the worksheet has unique headers and footers. options.DifferentFirst = true; // Set headers and footers for the first page. // Insert the workbook name into the header left section. options.FirstHeader.Left = "&F"; // Insert the current date into the footer left section. options.FirstFooter.Left = "&D"; // Insert the current page number into the footer right section. options.FirstFooter.Right = string.Format("Page {0} of {1}", "&P", "&N");</pre>	
Visual Basic	

```
Dim options As WorksheetHeaderFooterOptions = worksheet.HeaderFooterOptions
' Specify that the first page of the worksheet has unique headers and footers.
options.DifferentFirst = True
' Set headers and footers for the first page.
' Insert the workbook name into the header left section.
options.FirstHeader.Left = "&F"
' Insert the current date into the footer left section.
options.FirstFooter.Left = "&D"
' Insert the current page number into the footer right section.
options.FirstFooter.Right = String.Format("Page {0} of {1}", "&P", "&N")
```

The image below shows the result.



How to: Define a Print Area

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Printing](#) > [How to: Define a Print Area](#)

The table below lists a Worksheet object's methods that allow you to set or clear a **print area** on a worksheet. A print area designates one or more cell ranges as the only region to print. A worksheet can have multiple print areas. Each print area is printed on a separate page.

Method	Description
Worksheet.SetPrintRange	Defines a print area.
Worksheet.AddPrintRange	Adds cells to an existing print area.
Worksheet.ClearPrintRange	Clears a print area to print the entire worksheet.

C#

```
// Define a print area on a worksheet.
Range printArea = worksheet["B1:F39"];
worksheet.SetPrintRange(printArea);
// Enlarge the print area by adding adjacent cells.
// An additional print area is created if you add non-adjacent cells to the print area.
worksheet.AddPrintRange(worksheet["B40:F50"]);
// Uncomment the following line to clear the print area and print the entire worksheet.
// worksheet.ClearPrintRange();
```

Visual Basic

```
' Define a print area on a worksheet.
Dim printArea As Range = worksheet("B1:F39")
worksheet.SetPrintRange(printArea)
' Enlarge the print area by adding adjacent cells.
' An additional print area is created if you add non-adjacent cells to the print area.
worksheet.AddPrintRange(worksheet("B40:F50"))
' Uncomment the following line to clear the print area and print the entire worksheet.
' worksheet.ClearPrintRange();
```

How to: Print Titles on a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Printing](#) > [How to: Print Titles on a Worksheet](#)

This example demonstrates how to print titles (row and column headings) on every page. To do this, use the following members of the WorksheetPrintTitleOptions object, which can be accessed using the WorksheetPrintOptions.PrintTitles property.

Members	Description
WorksheetPrintTitleOptions.Rows WorksheetPrintTitleOptions.SetRows	Repeat specific rows at the top of every printed page.
WorksheetPrintTitleOptions.Columns WorksheetPrintTitleOptions.SetColumns	Repeat specific columns on the left side of every printed page.

C#

```
// Access an object that contains print options.  
WorksheetPrintOptions printOptions = worksheet.PrintOptions;  
// Print the first column and the first row on every page.  
printOptions.PrintTitles.SetColumns(0, 0);  
printOptions.PrintTitles.SetRows(0, 0);
```

Visual Basic

```
' Access an object that contains print options.  
Dim printOptions As WorksheetPrintOptions = worksheet.PrintOptions  
' Print the first column and the first row on every page.  
printOptions.PrintTitles.SetColumns(0, 0)  
printOptions.PrintTitles.SetRows(0, 0)
```

To remove print titles from a specific worksheet, use the WorksheetPrintTitleOptions.Clear method.

See Also
[How to: Print a Workbook](#)
[How to: Specify Print Settings](#)

How to: Use the WPF Chart Rendering Mechanism When Printing or Exporting

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Printing](#) > [How to: Use the WPF Chart Rendering Mechanism When Printing or Exporting a Workbook to PDF](#)

Charts are converted to images using the WinForms rendering mechanism when you print a workbook containing embedded charts or export it to PDF. However, if you use [Spreadsheet Document API](#) in a WPF application, you can use WPF charts when printing or exporting a document to PDF.

Follow the steps below to perform this task:

1. Add a reference to the **DevExpress.Xpf.Spreadsheet.v18.1.dll** assembly to your project.
2. Import the following namespaces into your code:

C#	
<pre>using DevExpress.Xpf.Spreadsheet.Services; using DevExpress.XtraSpreadsheet.Services; using DevExpress.XtraSpreadsheet.Services.Implementation;</pre>	
Visual Basic	
<pre>Imports DevExpress.Xpf.Spreadsheet.Services Imports DevExpress.XtraSpreadsheet.Services Imports DevExpress.XtraSpreadsheet.Services.Implementation</pre>	

3. Register the following services using the [Workbook.ReplaceService<T>](#) method to substitute the default chart rendering mechanism with the WPF one:
- **DevExpress.Xpf.Spreadsheet.Services.ChartControllerFactoryService**
 - **DevExpress.XtraSpreadsheet.Services.Implementation.WpfChartImageService**

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T603465>.

C#	
<pre>Workbook workbook = new Workbook(); // Replace the default chart rendering mechanism with the WPF one. workbook.ReplaceService<IChartControllerFactoryService>(new ChartControllerFactoryService()); workbook.ReplaceService<IChartImageService>(new WpfChartImageService()); workbook.LoadDocument("Document.xlsx"); // Export the workbook to PDF. workbook.ExportToPdf("ExportedDocument.pdf");</pre>	
Visual Basic	
<pre>Dim workbook As New Workbook() ' Replace the default chart rendering mechanism with the WPF one. workbook.ReplaceService(Of IChartControllerFactoryService)(New ChartControllerFactoryService()) workbook.ReplaceService(Of IChartImageService)(New WpfChartImageService()) workbook.LoadDocument("Document.xlsx") ' Export the workbook to PDF. workbook.ExportToPdf("ExportedDocument.pdf")</pre>	

Pictures

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pictures](#)

This section contains examples that demonstrate how to add pictures to a spreadsheet document.

- [How to: Insert and Delete Pictures](#)
- [How to: Modify an Embedded Picture](#)

How to: Insert and Delete Pictures

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pictures](#) > [How to: Insert and Delete Pictures](#)

The examples below demonstrate how to insert a picture into a worksheet from different sources using the `PictureCollection.AddPicture` method overloads. The `PictureCollection.GetPicturesByName` method enables you to get all pictures with the specified name; otherwise, you can find a picture by its unique ID using the `PictureCollection.GetPictureById` method. To remove a picture from a worksheet, use the `Shape.Delete` method of the `Picture` object.

From File or Stream

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(PictureActions.cs)
workbook.BeginUpdate();
// Set the measurement unit to Millimeter.
workbook.Unit = DevExpress.Office.DocumentUnit.Millimeter;
try
{
    Worksheet worksheet = workbook.Worksheets[0];
    // Insert a picture from a file so that its top left corner is in the specified cell.
    // By default the picture is named Picture 1.. Picture NN.
    worksheet.Pictures.AddPicture("Pictures\\x-docserver.png", worksheet.Cells["A1"]);
    // Insert a picture at 70 mm from the left, 40 mm from the top,
    // and resize it to a width of 85 mm and a height of 25 mm, locking the aspect ratio.
    worksheet.Pictures.AddPicture("Pictures\\x-docserver.png", 70, 40, 85, 25, true);
    // Insert the picture to be removed.
    worksheet.Pictures.AddPicture("Pictures\\x-docserver.png", 0, 0);
    // Remove the last inserted picture.
    // Find the shape by its name. The method returns a collection of shapes with the same name.
    Picture picShape = worksheet.Pictures.GetPicturesByName("Picture 3")[0];
    picShape.Delete();
}
finally
{
    workbook.EndUpdate();
}
```

Visual Basic

```
(PictureActions.vb)
workbook.BeginUpdate()
' Set the measurement unit to Millimeter.
workbook.Unit = DevExpress.Office.DocumentUnit.Millimeter
Try
    Dim worksheet As Worksheet = workbook.Worksheets(0)
    ' Insert a picture from a file so that its top left corner is in the specified cell.
    ' By default the picture is named Picture 1.. Picture NN.
    worksheet.Pictures.AddPicture("Pictures\\x-docserver.png", worksheet.Cells("A1"))
    ' Insert a picture at 70 mm from the left, 40 mm from the top,
    ' and resize it to a width of 85 mm and a height of 25 mm, locking the aspect ratio.
    worksheet.Pictures.AddPicture("Pictures\\x-docserver.png", 70, 40, 85, 25, True)
    ' Insert the picture to be removed.
    worksheet.Pictures.AddPicture("Pictures\\x-docserver.png", 0, 0)
    ' Remove the last inserted picture.
    ' Find the shape by its name. The method returns a collection of shapes with the same name.
    Dim picShape As Picture = worksheet.Pictures.GetPicturesByName("Picture 3")(0)
    picShape.Delete()
Finally
    workbook.EndUpdate()
End Try
```

From URI

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T2176> 15.

C#

```
(PictureActions.cs)
string imageUri = "https://www.devexpress.com/Products/NET/Controls/WinFor
// Create an image from Uri.
SpreadsheetImageSource imageSource = SpreadsheetImageSource.FromUri(imageU
// Set the measurement unit to point.
workbook.Unit = DevExpress.Office.DocumentUnit.Point;
workbook.BeginUpdate();
try
{
    Worksheet worksheet = workbook.Worksheets[0];
    // Insert a picture from the SpreadsheetImageSource at 100 pt from the
    // and resize it to a width of 300 pt and a height of 200 pt.
    worksheet.Pictures.AddPicture(imageSource, 100, 40, 300, 200);
}
finally
{
    workbook.EndUpdate();
}
```

Visual Basic

```
(PictureActions.vb)
Dim imageUri As String = "https://www.devexpress.com/Products/NET/Controls
' Create an image from Uri.
Dim imageSource As SpreadsheetImageSource = SpreadsheetImageSource.FromUri
' Set the measurement unit to point.
workbook.Unit = DevExpress.Office.DocumentUnit.Point
workbook.BeginUpdate()
Try
    Dim worksheet As Worksheet = workbook.Worksheets(0)
    ' Insert a picture from the SpreadsheetImageSource at 100 pt from the
    ' and resize it to a width of 300 pt and a height of 200 pt.
    worksheet.Pictures.AddPicture(imageSource, 100, 40, 300, 200)
Finally
    workbook.EndUpdate()
End Try
```

See Also

[How to: Modify an Embedded Picture](#)

How to: Modify an Embedded Picture

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Pictures](#) > [How to: Modify an Embedded Picture](#)

The example below demonstrates how to use various properties and methods of the Picture object to modify a picture inserted in a worksheet.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(PictureActions.cs)
// Set the measurement unit to Millimeter.
workbook.Unit = DevExpress.Office.DocumentUnit.Millimeter;
workbook.BeginUpdate();
try
{
    Worksheet worksheet = workbook.Worksheets[0];
    // Insert pictures from the file.
    Picture pic = worksheet.Pictures.AddPicture("Pictures\\x-docserver.png", worksheet.Cells["A1"]);
    // Specify picture name and draw a border.
    pic.Name = "Logo";
    pic.AlternativeText = "Spreadsheet Logo";
    pic.BorderWidth = 1;
    pic.BorderColor = DevExpress.Utils.DXColor.Black;
    // Move a picture.
    pic.Move(20, 30);
    // Change picture behavior so it will move and size with underlying cells.
    pic.Placement = Placement.MoveAndSize;
    worksheet.Rows[5].Height += 10;
    worksheet.Columns["D"].Width += 10;
    // Specify rotation angle.
    pic.Rotation = 30;
    // Add a hyperlink.
    pic.InsertHyperlink("http://www.devexpress.com/Products/NET/Document-Server/", true);
}
finally
{
    workbook.EndUpdate();
}
```

Visual Basic

```
(PictureActions.vb)
' Set the measurement unit to Millimeter.
workbook.Unit = DevExpress.Office.DocumentUnit.Millimeter
workbook.BeginUpdate()
Try
    Dim worksheet As Worksheet = workbook.Worksheets(0)
    ' Insert pictures from the file.
    Dim pic As Picture = worksheet.Pictures.AddPicture("Pictures\x-docserv
    ' Specify picture name and draw a border.
    pic.Name = "Logo"
    pic.AlternativeText = "Spreadsheet Logo"
    pic.BorderWidth = 1
    pic.BorderColor = DevExpress.Utils.DXColor.Black
    ' Move a picture.
    pic.Move(20, 30)
    ' Change picture behavior so it will move and size with underlying cell.
    pic.Placement = Placement.MoveAndSize
    worksheet.Rows(5).Height += 10
    worksheet.Columns("D").Width += 10
    ' Specify rotation angle.
    pic.Rotation = 30
    ' Add a hyperlink.
    pic.InsertHyperlink("http://www.devexpress.com/Products/NET/Document-S
Finally
    workbook.EndUpdate()
End Try
```

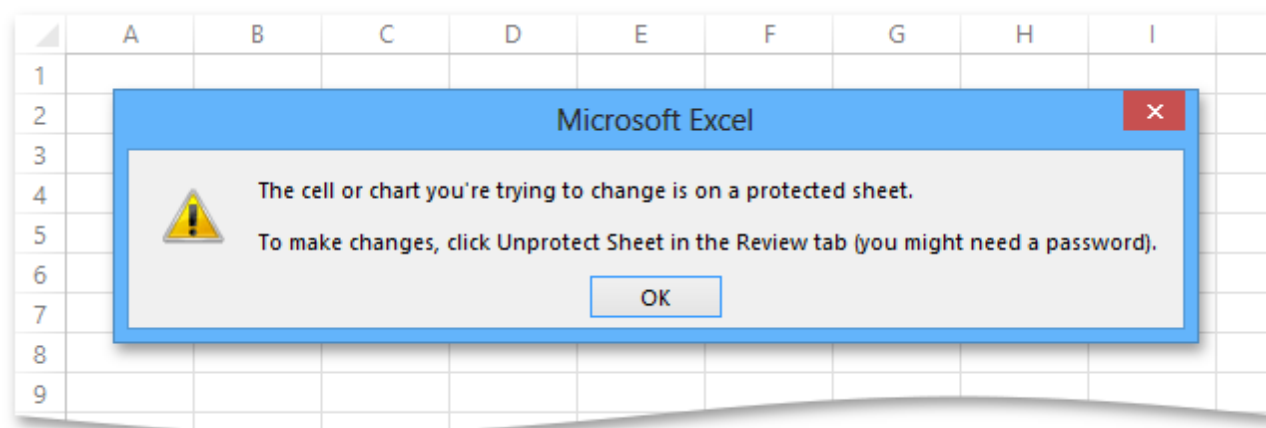
See Also

[How to: Insert and Delete Pictures](#)

Protection

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Protection](#)

This section contains examples that demonstrate how to impose document protection to prevent end-users from modifying workbook structure, individual worksheets, and specific ranges.



- [How to: Protect a Workbook](#)
- [How to: Protect a Worksheet](#)
- [How to: Protect Specific Worksheet Ranges](#)

How to: Protect a Workbook

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Protection](#) > [How to: Protect a Workbook](#)

The example below demonstrates how to protect an entire workbook by using the [Workbook.Protect](#) method. Workbook protection prevents end-users from modifying a workbook structure (by moving, deleting, adding, renaming, or hiding existing worksheets, or displaying hidden sheets).

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(ProtectionActions.cs)
Worksheet worksheet = workbook.Worksheets["ProtectionSample"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Protect workbook structure with the password (prevent users from adding or
//deleting worksheets or displaying hidden worksheets).
if (!workbook.IsProtected)
    workbook.Protect("password", true, false);
// Add a note.
worksheet["B2"].Value = "Workbook structure is protected with a password. \n You cannot add, move or delete
worksheet.Visible = true;
```

Visual Basic

```
(ProtectionActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("ProtectionSample")
workbook.Worksheets.ActiveWorksheet = worksheet
' Protect workbook structure with the password (prevent users from adding
'deleting worksheets or displaying hidden worksheets).
If Not workbook.IsProtected Then
    workbook.Protect("password", True, False)
End If
' Add a note.
worksheet("B2").Value = "Workbook structure is protected with a password.
worksheet.Visible = True
```

Unprotect a workbook

To remove protection, use the [Workbook.Unprotect](#) method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(ProtectionActions.cs)
Worksheet worksheet = workbook.Worksheets["ProtectionSample"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Unprotect the workbook using a password.
if (workbook.IsProtected)
    workbook.Unprotect("password");
// Add a note.
worksheet["B2"].Value = "Workbook is unprotected. Worksheets can be added,
worksheet.Visible = true;
```

Visual Basic

```
(ProtectionActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("ProtectionSample")
workbook.Worksheets.ActiveWorksheet = worksheet
' Unprotect the workbook using a password.
If workbook.IsProtected Then
    workbook.Unprotect("password")
End If
' Add a note.
worksheet("B2").Value = "Workbook is unprotected. Worksheets can be added,
worksheet.Visible = True
```

See Also

[How to: Protect a Worksheet](#)

[How to: Protect Specific Worksheet Ranges](#)

How to: Protect a Worksheet

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Protection](#) > [How to: Protect a Worksheet](#)

You can protect a worksheet by locking cells, so that end-users can only perform a specific (restricted) set of actions specified by the WorksheetProtectionPermissions enumeration member.

By default, all cells in a worksheet have the Protection.Locked attribute set to **true**. When protection is applied to a worksheet, these cells become read-only. Cells with the Protection.Locked attribute set to **false** are not protected when worksheet protection is applied, so that the end-users can edit or delete their content.

To apply worksheet protection, use the Worksheet.Protect method, as illustrated in the example below. By default, when you pass the WorksheetProtectionPermissions.Default parameter to the **Protect** method, a protected worksheet is locked so that an end-user can only select cells.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(ProtectionActions.cs)
Worksheet worksheet = workbook.Worksheets["ProtectionSample"];
workbook.Worksheets.ActiveWorksheet = worksheet;
// Protect the worksheet. Prevent end-users from changing worksheet elements.
if (!worksheet.IsProtected)
    worksheet.Protect("password", WorksheetProtectionPermissions.Default);
// Add a note.
worksheet["B2"].Value = "Worksheet is protected with a password. \n You cannot edit or format cells until \n\nTo remove protection, on the Review tab, in the Changes group," +
    "\nclick \"Unprotect Sheet\" and enter \"password\".";
worksheet.Visible = true;
```

Visual Basic

```
(ProtectionActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("ProtectionSample")
workbook.Worksheets.ActiveWorksheet = worksheet
' Protect the worksheet. Prevent end-users from changing worksheet element
If Not worksheet.IsProtected Then
    worksheet.Protect("password", WorksheetProtectionPermissions.Default)
End If
' Add a note.
worksheet("B2").Value = "Worksheet is protected with a password. " & Contr
worksheet.Visible = True
```

Unprotect a worksheet

To remove protection, use the Worksheet.Unprotect method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

Visual Basic

```
(ProtectionActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("ProtectionSample")
workbook.Worksheets.ActiveWorksheet = worksheet
' Remove worksheet protection using a password.
If worksheet.IsProtected Then
    worksheet.Unprotect("password")
End If
' Add a note.
worksheet("B2").Value = "Worksheet is unprotected. You can edit and format"
worksheet.Visible = True
```

See Also

[How to: Protect a Workbook](#)

[How to: Protect Specific Worksheet Ranges](#)

How to: Protect Specific Worksheet Ranges

[Office File API](#) > [Spreadsheet Document API](#) > [Examples](#) > [Protection](#) > [How to: Protect Specific Worksheet Ranges](#)

This example demonstrates how to unlock specific ranges in a protected worksheet for authenticated users.

The ProtectedRangeCollection collection (accessible using the Worksheet.ProtectedRanges property) contains ranges that can be unlocked for users providing the required credentials. A user can be authenticated in one of the following ways.

- Authentication that uses the user account under which the application runs. The authentication itself is performed by Windows based on the standard security mechanisms that rely on domain security.

To unlock a range for a specific user or user group, perform the following steps.

1. Create an `EditRangePermission` object for each user or group.
 2. Transform a list of permissions to a security descriptor by using the `ProtectedRange.CreateSecurityDescriptor` method.
 3. Assign the created security descriptor to the range by using the `ProtectedRange.SecurityDescriptor` property.
- Authentication that uses a password. To specify a password for the range, use the `ProtectedRange.SetPassword` method. When an attempt is made to edit the range, the user will be prompted for a password.

The code example below illustrates both approaches.

 Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T217615>.

C#

```
(ProtectionActions.cs)
Worksheet worksheet = workbook.Worksheets["ProtectionSample"];
workbook.Worksheets.ActiveWorksheet = worksheet;
worksheet["B2:J5"].Borders.SetOutsideBorders(Color.Red, BorderLineStyle.Thin);
// Specify user permission to edit a range in a protected worksheet.
ProtectedRange protectedRange = worksheet.ProtectedRanges.Add("My Range", worksheet["B2:J5"]);
EditRangePermission permission = new EditRangePermission();
permission.UserName = Environment.UserName;
permission.DomainName = Environment.UserDomainName;
permission.Deny = false;
protectedRange.SecurityDescriptor = protectedRange.CreateSecurityDescriptor(new EditRangePermission[] { permission });
protectedRange.SetPassword("123");
// Protect a worksheet.
if (!worksheet.IsProtected)
    worksheet.Protect("password", WorksheetProtectionPermissions.Default);
// Add a note.
worksheet["B2"].Value = "This cell range is protected with a password. \n You cannot edit or format it until the password is removed. \n\nTo remove protection, double-click the range and enter \"123\".";
worksheet.Visible = true;
```

Visual Basic

```
(ProtectionActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets("ProtectionSample")
workbook.Worksheets.ActiveWorksheet = worksheet
worksheet("B2:J5").Borders.SetOutsideBorders(Color.Red, BorderLineStyle.Th
' Specify user permission to edit a range in a protected worksheet.
Dim protectedRange As ProtectedRange = worksheet.ProtectedRanges.Add("My R
Dim permission As New EditRangePermission()
permission.UserName = Environment.UserName
permission.DomainName = Environment.UserDomainName
permission.Deny = False
protectedRange.SecurityDescriptor = protectedRange.CreateSecurityDescriptor
protectedRange.SetPassword("123")
' Protect a worksheet.
If Not worksheet.IsProtected Then
    worksheet.Protect("password", WorksheetProtectionPermissions.Default)
End If
' Add a note.
worksheet("B2").Value = "This cell range is protected with a password. " &
worksheet.Visible = True
```

See Also

[How to: Protect a Workbook](#)

[How to: Protect a Worksheet](#)

Word Processing Document API

[Office File API](#) > [Word Processing Document API](#)

The Word Processing Document API is a non-visual .NET library which allows you to automate frequent word processing tasks (format conversion, document editing in code, mail merge, printing to PDF, etc.).

Note

The Word Processing Document API does not require the DevExpress.Docs.v18.1.dll assembly for operation and distribution. A separate license is not required.

The Word (RTF) Document API's main features are listed below.

Import and Export

Create, [load](#), [convert](#) and [save](#) documents to popular file formats:

- Plain Text
- RTF (Rich Text Format)
- DOCX (Microsoft Office 2007 - ... format)
- DOC (Microsoft Word 97-2003 format)
- WordML format (MS Office Word 2003 XML format)
- OpenDocument (implemented by the OpenOffice.org office suite)
- HTML
- MHTML (web page archive format)
- EPUB (Electronic Publication)
- PDF format (export only)

Document Basics

- [Combine](#) multiple documents into a single file.
- Select a specific document part (text range, paragraph, etc.), change its format, or extract it as a separate document.
- [Print](#) documents with the default or custom printing settings.

Text Formatting Features

- Format [characters](#) by changing the font, font size, character style (bold, italics, underlined and strike-through), and different background and foreground colors.

[Paragraphs](#) formatting options include alignment, indentation, variable paragraph, and line spacing.

- Create and modify [bulleted, numbered and multi-level lists](#).
- Apply paragraph and character-based [styles](#).
- Insert inline [images](#) using popular formats such as JPEG, JPG, PNG, GIF, BMP, TIF, TIFF, WMF, EMF, and DIB.
- Create, move, resize and rotate [floating objects](#).
- Insert and modify [hyperlinks and bookmarks](#).
- Create and adjust [checkboxes](#).
- Perform [table operations](#) such as inserting tables, rows and columns, editing the table layout by splitting and merging cells, aligning cell content, applying borders and shading to individual cells, and deleting cells, columns and rows.

Mail Merge

- Create a document template for letters, catalogs, mailing labels and personalize any kind of document using the [mail merge](#) feature.
- Nest master and detail templates within the basic merge template to create a [master-detail report](#). Use this feature to create catalogs, header/detail invoices or statements.

Field Support

- The Word Processing Document API supports a subset of fields defined in the ECMA-376 (Office OpenXML) standard, including the [DOCVARIABLE](#) field.

Document Layout

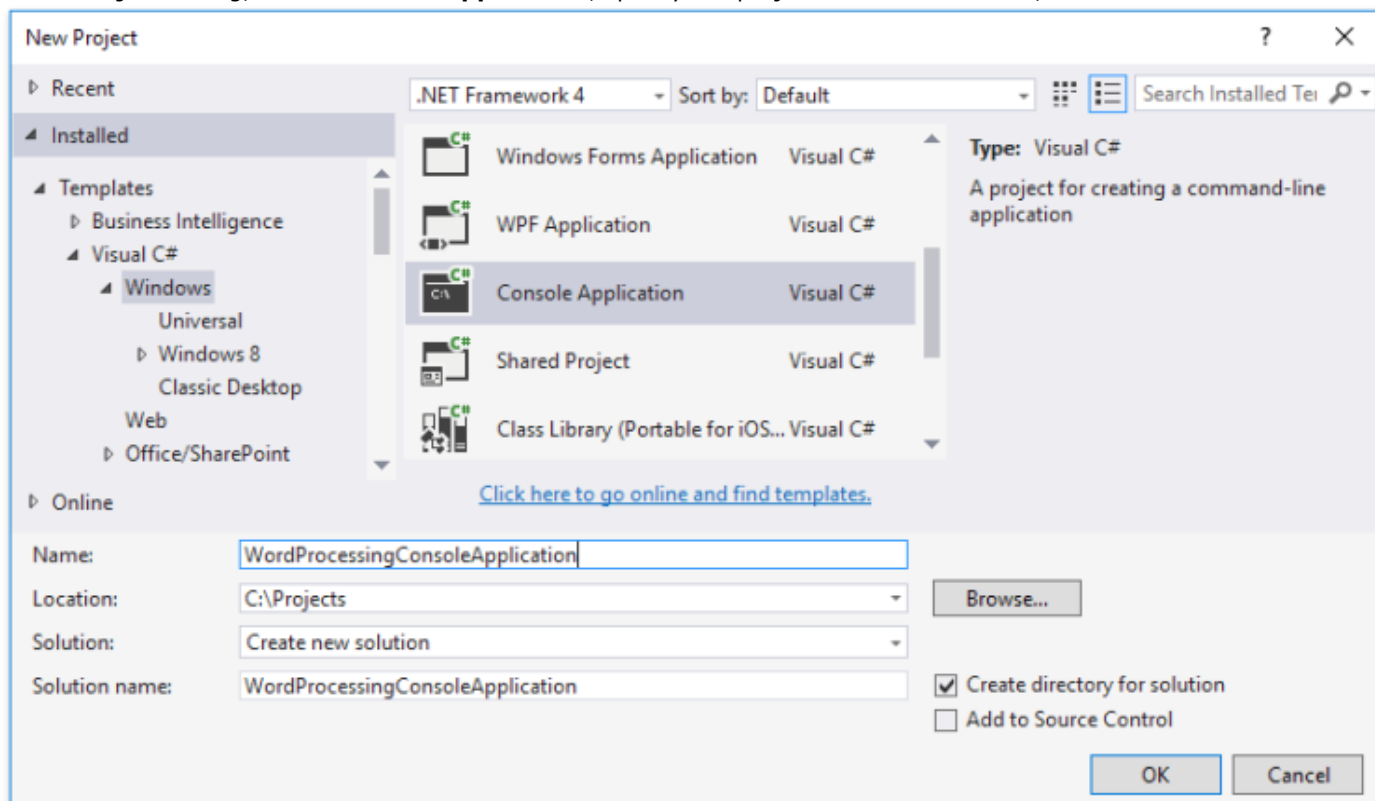
- Parse the document into different [sections](#) with individual page settings.
- Split text in multiple columns within a particular section.
- Specify different [headers and footers](#) for the first page, odd and even pages, and each section. Insert the current page number and the total number of pages.
- Add [line numbers](#) to document margins for certain types of legal documents. Line numbers can run continuously throughout the document, restart on each page or section, or hidden be suppressed for a specific paragraph.

Getting Started

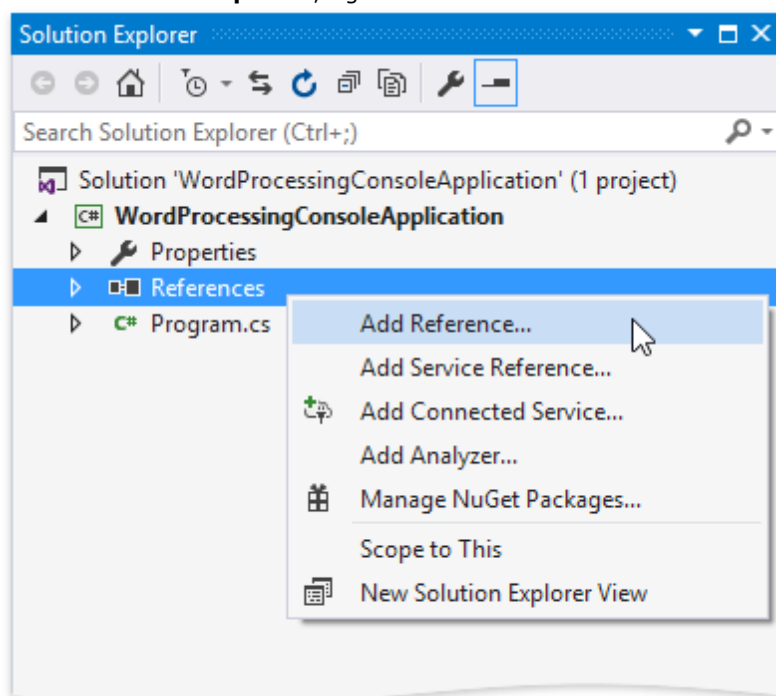
[Office File API](#) > [Word Processing Document API](#) > [Getting Started](#)

This tutorial describes how to get started with the Word Processing Document API.

1. Start Visual Studio and create a new project by selecting **FILE | New | Project** in the main menu. In the invoked **New Project** dialog, select **Console Application**, specify the project name and location, and click **OK**.



2. In the **Solution Explorer**, right-click the **References** node and select **Add Reference...** in the context menu.



3. In the invoked **Reference Manager** dialog, add references to the following libraries.

- System.Drawing.dll (to follow this example only)
 - DevExpress.Data.v18.1.dll
 - DevExpress.Office.v18.1.Core.dll
 - DevExpress.RichEdit.v18.1.Core.dll
 - DevExpress.Printing.v18.1.Core.dll
4. Paste the code listed below in the **Main** method of the *Program.cs* file (or the **Main** procedure of the *Module1.vb* file for Visual Basic). This example creates and saves the *Test.docx* file, and opens it in the default application registered for this file type.

C#

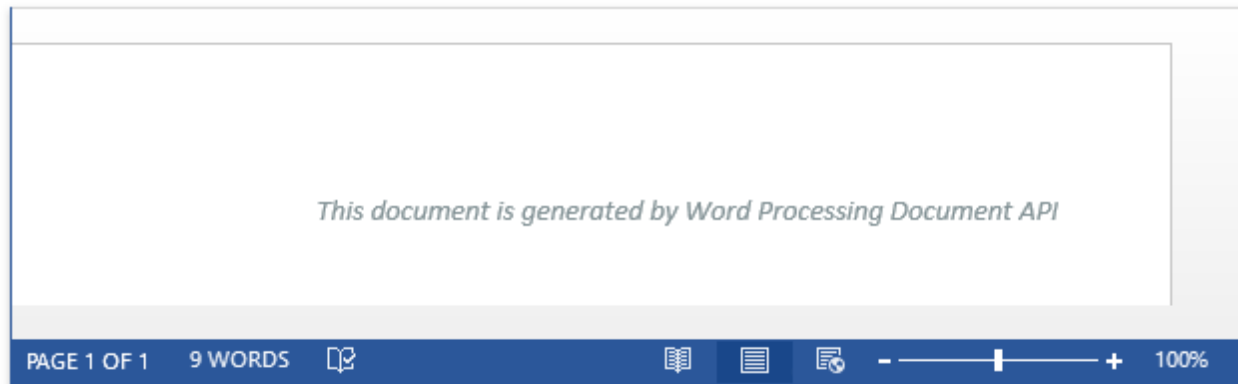
```
static void Main(string[] args)
{
    string fileName = "Test.docx";
    using (DevExpress.XtraRichEdit.RichEditDocumentServer srv =
        new DevExpress.XtraRichEdit.RichEditDocumentServer())
    {
        DevExpress.XtraRichEdit.API.Native.Document doc = srv.Document;
        doc.AppendText("This document is generated by Word Processing Document API");
        DevExpress.XtraRichEdit.API.Native.CharacterProperties cp = doc.BeginUpdateCharacters(doc.Paragraph
        cp.ForeColor = System.Drawing.Color.FromArgb(0x83, 0x92, 0x96);
        cp.Italic = true;
        doc.EndUpdateCharacters(cp);
        DevExpress.XtraRichEdit.API.Native.ParagraphProperties pp = doc.BeginUpdateParagraphs(doc.Paragraph
        pp.Alignment = DevExpress.XtraRichEdit.API.Native.ParagraphAlignment.Right;
        doc.EndUpdateParagraphs(pp);
        srv.SaveDocument(fileName, DevExpress.XtraRichEdit.DocumentFormat.OpenXml);
    }
    System.Diagnostics.Process.Start(fileName);
}
```

Visual Basic

```
Shared Sub Main(ByVal args() As String)
    Dim fileName As String = "Test.docx"
    Using srv As New DevExpress.XtraRichEdit.RichEditDocumentServer()
        Dim doc As DevExpress.XtraRichEdit.API.Native.Document = srv.Document
        doc.AppendText("This document is generated by Word Processing Document API")
        Dim cp As DevExpress.XtraRichEdit.API.Native.CharacterProperties = doc.BeginUpdateCharacters(doc.I
        cp.ForeColor = System.Drawing.Color.FromArgb(&H83, &H92, &H96)
        cp.Italic = True
        doc.EndUpdateCharacters(cp)
        Dim pp As DevExpress.XtraRichEdit.API.Native.ParagraphProperties = doc.BeginUpdateParagraphs(doc.I
        pp.Alignment = DevExpress.XtraRichEdit.API.Native.ParagraphAlignment.Right
        doc.EndUpdateParagraphs(pp)
        srv.SaveDocument(fileName, DevExpress.XtraRichEdit.DocumentFormat.OpenXml)
    End Using
    System.Diagnostics.Process.Start(fileName)
End Sub
```

5. Run the project.

The following image demonstrates the file generated after executing the code above (the document is opened in Microsoft® Word®).



Word Processing Document

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#)

The topics in this section give you basic information on Word Processing Document API structure.

The following sections are available:

- [Document Structure](#)
- [Document Elements](#)

Document Structure

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Structure](#)

WordProcessing Document API defines two types of structures for a document. First, the logical structure of a document, named **Document Model**, which reflects its logical hierarchical organization. Second, the physical structure, which represents the appearance of physical entities in a document. This is called the **Document Layout**, since it is a hierarchical description of a layout of a document.

When a document is processed by RichEditDocumentServer, the Document Model is initially built. Then the **Layout Engine** converts the Document Model into the Document Layout. So in general, there is no one to one correspondence between physical and logical entities.

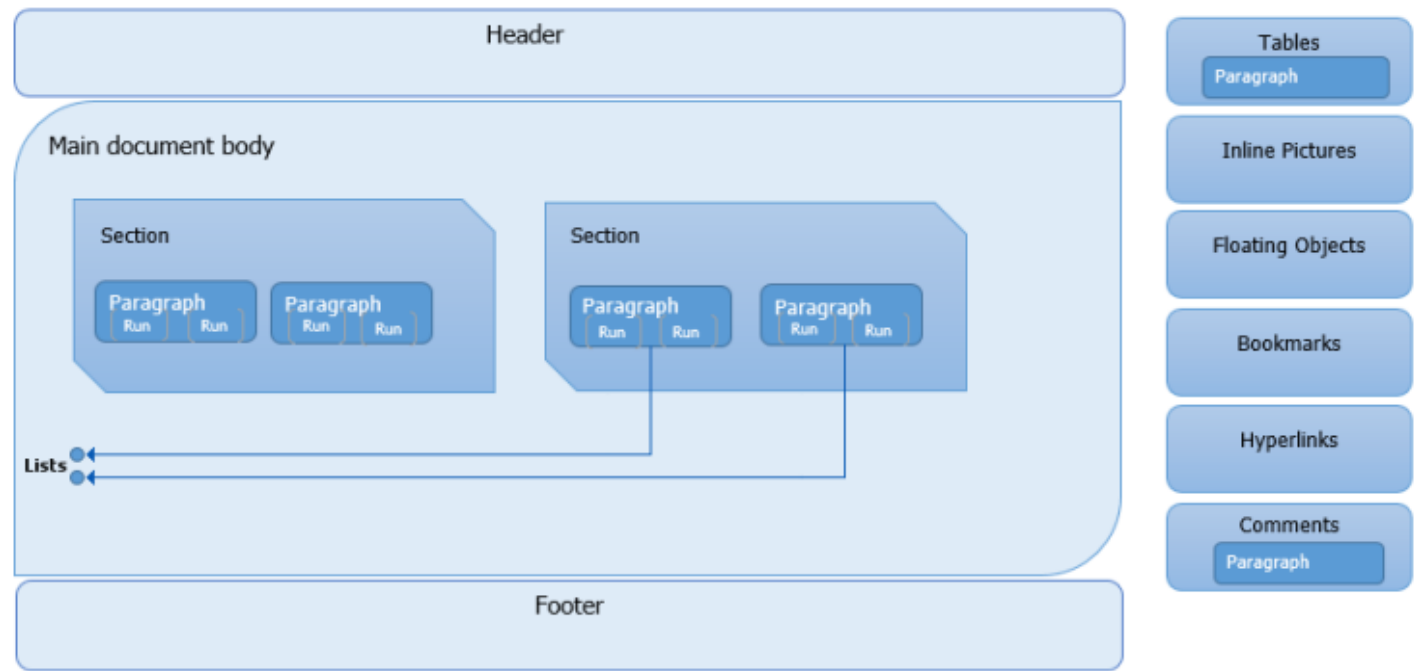
The following articles describe document structures in more detail:

- [Document Model](#)
- [Document Layout](#)

Document Model

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Structure](#) > [Document Model](#)

A document model defines the logical structure of a document reflecting its logical organization and hierarchically groups the logical entities in a document. It is illustrated in the following diagram.



The root of the document model is the object exposing the SubDocument interface. It delivers the basic document functionality that is common for the main document body, header and footer. Specific functionality belongs to its descendant, the Document interface. Use the RichEditControl.Document property to get access to the **Document** interface.

The document is divided into logical Document.Sections. Each section in a SectionCollection defines its own page settings, such as paper size, page orientation and margins. The SectionPage.PaperKind, SectionPage.Landscape, Section.Page and Section.Margins properties enable you to access these settings.

To insert a section into an arbitrary position, use the Document.InsertSection method.

The **section** consists of **paragraphs**. The Paragraph class represents the **paragraph** entity in the document model. The Section.Paragraphs property of a **section** gives you a collection of paragraphs in a particular section of a document. If you need to access the collection of paragraphs in a whole document, use the SubDocument.Paragraphs property.

To modify paragraph formatting, you can use either the Paragraph object or the ParagraphProperties interface. The interface is more convenient to use, since it can be easily accessed for a particular document range via the SubDocument.BeginUpdateParagraphs method. Do not forget to call the SubDocument.EndUpdateParagraphs method to finish the modification appropriately.

To insert a paragraph at an arbitrary position, use the ParagraphCollection.Insert method.

The paragraph consists of **runs** - groups of characters with a uniform format. This is an internal entity, not available via API. Manage the Positions and Ranges instead.

Note

For the specified DocumentPosition, use the SubDocument.GetParagraph method to get an encompassing Paragraph, use the Document.GetSection to get an encompassing Section.

The document model also includes entities that do not comprise the hierarchy, such as **lists**, **styles**, **tables**, **inline pictures**, **floating objects** (pictures and text boxes), **bookmarks**, **hyperlinks** and **comments**.

Object	Description	Collection containing an object:	Exposed By
--------	-------------	----------------------------------	------------

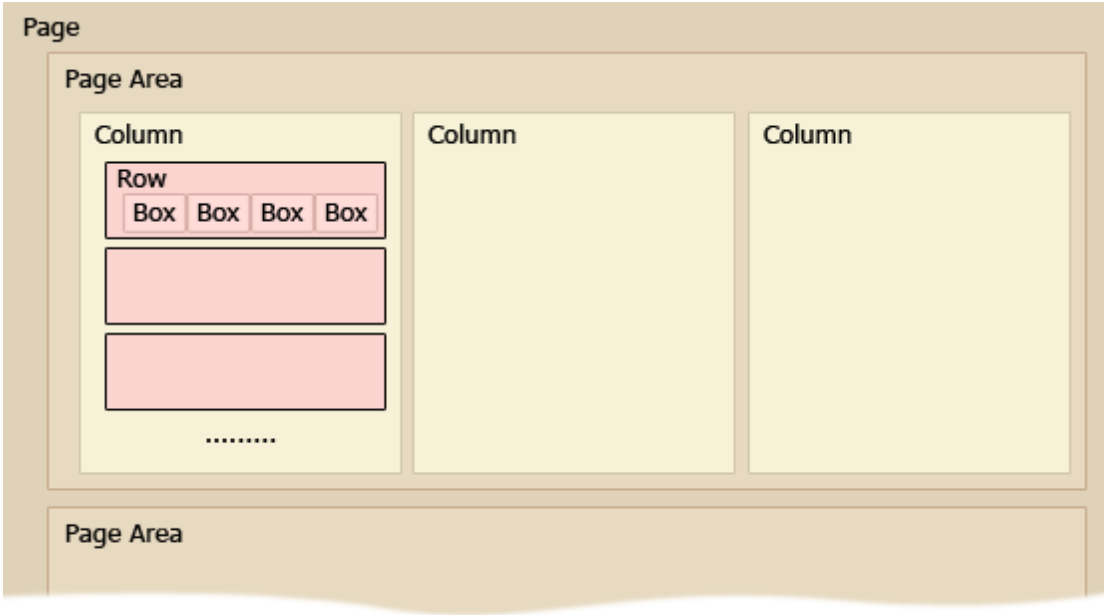
Lists	Represented by a specific collection of paragraphs. This collection is composed of items with a NumberingList interface. Each item is an instance of a certain abstract numbering definition, which is represented by an AbstractNumberingList interface.	NumberingListCollection AbstractNumberingListCollection	Document.AbstractNumberingLists NumberingList.AbstractNumberingList ParagraphCollection.AddParagraphsToList
ParagraphStyle CharacterStyle	Pertains to a document and belongs to a particular document model. You cannot add the style object from a style collection of one document to the style collection of another document.	ParagraphStyleCollection CharacterStyleCollection	Document.CharacterStyles Document.ParagraphStyles CharacterStyleCollection.CreateNew ParagraphStyleCollection.CreateNew
Table	Specific structure based on paragraphs. To change the cell content, you should operate with a corresponding paragraph.	TableCollection ReadOnlyTableCollection	SubDocument.Tables TableCollection.Create ReadOnlyTableCollection.Get
DocumentImage	Represents the inline picture entity in the document model. Occupies the range equivalent to one character, i.e., the DocumentRange.Length of the range is equal to 1.	DocumentImageCollection ReadOnlyDocumentImageCollection	DocumentImage.Image DocumentImageCollection.Insert DocumentImageCollection.Append ReadOnlyDocumentImageCollection.Get
Shape	Represents floating object (pictures and text boxes) entities in the document model. Anchored to the position, but can be placed anywhere on the page that contains the anchor - within, above and below the main document body.	ShapeCollection ReadOnlyShapeCollection	SubDocument.Shapes Shape.TextBox ReadOnlyShapeCollection.Get
Bookmark	Represents bookmark entity in the document model. Linked to the position, so the range can have a zero length.	BookmarkCollection ReadOnlyBookmarkCollection	SubDocument.Bookmarks BookmarkCollection.Create ReadOnlyBookmarkCollection.Get
Hyperlink	Represents a hyperlink entity in the document model. Can be linked to the bookmark.	HyperlinkCollection ReadOnlyHyperlinkCollection	SubDocument.Hyperlinks HyperlinkCollection.Create ReadOnlyHyperlinkCollection.Get
Comment	Represents a comment entity in the document model. Linked to the document range. Can be nested - one comment can relate to another. In this case, only the parent comment is linked to the	CommentCollection ReadOnlyCommentCollection	SubDocument.Comments Comment.ParentComment ReadOnlyCommentCollection.Get

	document range, whereas the child comment is linked to its parent.		
--	--	--	--

Document Layout

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Structure](#) > [Document Layout](#)

The document layout represents the physical model of the document, constructed by the *Layout Engine* from the logical Document Model. The hierarchical structure of the document layout is illustrated in the picture below.



Page settings are specified for each document section via the `Section.Page` and `Section.Margins` properties. The `Section.Columns` property provides access to columns defined in the current section via the `SectionColumns` interface. To divide a section into columns, use the `SectionColumns.CreateUniformColumns` method to create a columns layout and the `SectionColumns.SetColumns` method to apply the layout.

The visual appearance of text characters, especially at small font sizes, is dependent on the units of measurement used for internal calculations when symbols are rendered. Rounding errors may lead to font artifacts, irregular font spacing and other minor glitches. Change the `RichEditControl.LayoutUnit` property value to `DocumentLayoutUnit.Pixel` to improve the appearance of the document.

Document Elements

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#)

The Document and the SubDocument interfaces are the entry points for programmatic access and modification of the document. Refer to the **DevExpress.XtraRichEdit.API.Native** topic which lists the most important entities of the Word (RTF) Document API.

The following articles describe document structural elements in more detail:

- [Positions and Ranges](#)
- [Characters](#)
- [Paragraphs](#)
- [Hyperlinks and Bookmarks](#)
- [Headers and Footers](#)
- [Tables](#)
- [Sections](#)
- [Styles](#)
- [Numbered and Bulleted Lists](#)

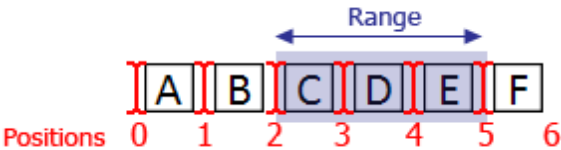
Positions and Ranges

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Positions and Ranges](#)

Position and range are cornerstone Rich Edit entities that define the location of every object within a logical document structure.

Position is a zero-based index of every character within a text string. It is represented by the DocumentPosition class. The best visual presentation of the position is a text cursor (caret). Placed in the beginning of the document, it is located before the first character. This is a 0 position. When the cursor is moved forward to the next character, it moves to the position 1, and so on.

The *range* in turn is a distance between two positions, represented by the DocumentRange.Start and DocumentRange.End properties. The difference between the ending and starting position defines the DocumentRange.Length.



To create a DocumentRange, use the SubDocument.CreateRange method. To obtain a range of the selected document fragment, use the Document.Selection property.

The text can be added to the range by the SubDocument.AppendText, SubDocument.InsertText or SubDocument.InsertSingleLineText methods. Adding new text elements increases the DocumentRange.End and DocumentRange.Length property values.

To delete the characters from the range, use the SubDocument.Delete method. After deletion, the DocumentRange.End and DocumentRange.Length property values will lessen.

If the text preceding the range is removed, the DocumentRange.Length value remains the same, whereas the start and end range positions move backward, hence the DocumentRange.Start and DocumentRange.End values decrease.

Each document entity has the **Range** property, so you can get the range occupied by the specific unit. And vice versa, you can retrieve the document entity from the specified range. The following table shows what method should be used to retrieve any document entity.

Object	Method
Text	SubDocument.GetText SubDocument.GetRtfText SubDocument.GetHtmlText SubDocument.GetMhtText SubDocument.GetOpenXmlBytes
Paragraph	ReadOnlyParagraphCollection.Get
Section	Document.GetSection
Image	ReadOnlyDocumentImageCollection.Get
Shape (floating image or text box)	ReadOnlyShapeCollection.Get
Table	ReadOnlyTableCollection.Get
Hyperlink	ReadOnlyHyperlinkCollection.Get
Bookmark	ReadOnlyBookmarkCollection.Get
Comment	ReadOnlyCommentCollection.Get

Field	ReadOnlyFieldCollection.Get
-------	-----------------------------

Note

If you operate with a selection range, any of the above-listed methods should be enclosed within DocumentRange.BeginUpdateDocument - DocumentRange.EndUpdateDocument methods pair. Otherwise, an incorrect document model might be selected, resulting in an exception **"Error: specified document position or range belongs to other document or subdocument"** being thrown.

Since there is no exact correspondence between the logical and physical document structures, it is hard to determine a line or a page where a certain range or position is located. But there are a few methods that can help you obtain an approximate location of the required position.

- To obtain coordinates of the rectangle surrounding the specified document position, use the RichEditControl.GetBoundsFromPosition method.
- The RichEditControl.GetPositionFromPoint method enables you to locate the position, which is situated close to the specified point.
- The RichEditView.GetCursorBounds provides coordinates of a rectangle encompassing the cursor.

Characters

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Characters](#)

The RichEdit API implements several methods used to retrieve the content of the specified range in different formats. These methods are as follows:

- SubDocument.GetText
- SubDocument.GetRtfText
- SubDocument.GetHtmlText
- SubDocument.GetMhtText
- SubDocument.GetWordMLText
- SubDocument.GetOpenXmlBytes

Characters can be formatted using different settings for font, font size and character style (e.g., bold, italics, underlined and strike-through), and different colors for background and foreground. To modify the formatting of characters in a range, use the SubDocument.BeginUpdateCharacters - SubDocument.EndUpdateCharacters method pair. This method pair provides access to the CharacterProperties interface, which enables you to either get or change character formatting.

The CharacterProperties.Style property allows you to apply a character style to a range.

Paragraphs

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Paragraphs](#)

The Section document consists of **paragraphs**. The Paragraph class represents the **paragraph** entity in the document model. You can access the collection of paragraphs in a document via the SubDocument.Paragraphs property. The Section.Paragraphs property provides access to a collection of paragraphs in a particular section of a document.

To insert a paragraph in an arbitrary position, use the SubDocument.InsertParagraph method.

For the specified DocumentPosition, use the ReadOnlyParagraphCollection.Get method (**Document.Paragraphs.Get(DocumentRange range)** notation) to get the encompassing Paragraph.

Paragraph formatting can be modified using the ParagraphProperties object, which is accessible via the SubDocument.BeginUpdateParagraphs method.

To modify formatting for a paragraph that encompasses the specified range, get the SubDocument interface for this range first. To do this, you can use the DocumentPosition.BeginUpdateDocument - DocumentPosition.EndUpdateDocument method pair or the DocumentRange.BeginUpdateDocument - DocumentRange.EndUpdateDocument method pair. Then use the SubDocument.BeginUpdateParagraphs - SubDocument.EndUpdateParagraphs method pair. The SubDocument.BeginUpdateParagraphs method requires a DocumentRange as the parameter, and returns a ParagraphProperties interface that exposes formatting properties for this range.

To modify text of the paragraph, obtain its Paragraph.Range or any position(s) within that range and use SubDocument methods (for example, SubDocument.InsertRtfText, SubDocument.AppendText etc.) for the obtained DocumentRange object.

Pictures

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Pictures](#)

Pictures in a document can be inline or floating objects.

Note

This document describes inline pictures. For information about floating pictures, refer to the [Floating Objects](#) article.

Inline picture in a document is represented by the `DocumentImage` object. It occupies the range equivalent to one character, i.e. the `DocumentRange.Length` of the range is equal to 1.

The `DocumentImage.Image` property provides access to the previous level of image abstraction, and enables you to get information on image size, resolution and color depth. The `OfficeImage` object obtained via this property can be used to get a native .NET [Image](#) (the `OfficeImage.NativeImage` property) or to get image data in a different format (the `OfficeImage.GetImageBytes` method).

All images in the document are contained in the `DocumentImageCollection` accessible using the `SubDocument.Images` property. Use the `DocumentImageCollection.Insert` and `DocumentImageCollection.Append` methods to insert pictures into a document.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4803>.

This example illustrates how to insert inline pictures in a rich text document using the `RichEditDocumentServer`. The image to insert can be obtained from a file, from a stream or from a URI.

C#

```
(Program.cs)
using System.IO;
using System.Reflection;
using DevExpress.XtraRichEdit;
using DevExpress.XtraRichEdit.API.Native;

RichEditDocumentServer server = new RichEditDocumentServer();
server.LoadDocument("Texts\\InlinePictures.rtf", DocumentFormat.Rtf);
Document doc = server.Document;
// Insert an image from a file.
DocumentRange rangeFound = doc.FindAll("Visual Studio Magazine", SearchOptions.CaseSensitive)
DocumentPosition pos = doc.Paragraphs[doc.Paragraphs.Get(rangeFound.End).Index + 2].Range.Start;
doc.Images.Insert(pos, DocumentImageSource.FromFile("Pictures\\ReadersChoice.png"));
// Insert an image from a stream.
pos = doc.Paragraphs[4].Range.Start;
string imageToInsert = "information.png";
Assembly a = Assembly.GetExecutingAssembly();
Stream imageStream = a.GetManifestResourceStream("InlinePictures.Resources." + imageToInsert);
doc.Images.Insert(pos, DocumentImageSource.FromStream(imageStream));
// Insert an image using its URI.
string imageUri = "http://i.gyazo.com/798a2ed48a3535c6c8add0ea7a4fc4e6.png";
SubDocument docHeader = doc.Sections[0].BeginUpdateHeader();
docHeader.Images.Append(DocumentImageSource.FromUri(imageUri, server));
doc.Sections[0].EndUpdateHeader(docHeader);
// Insert a barcode.
DevExpress.BarCodes.BarCode barCode = new DevExpress.BarCodes.BarCode();
barCode.Symbology = DevExpress.BarCodes.Symbology.QRCode;
barCode.CodeText = "http://www.devexpress.com";
barCode.CodeBinaryData = System.Text.Encoding.Default.GetBytes(barCode.CodeText);
barCode.Module = 0.5;
SubDocument docFooter = doc.Sections[0].BeginUpdateFooter();
docFooter.Images.Append(barCode.BarCodeImage);
doc.Sections[0].EndUpdateFooter(docFooter);
```

Visual Basic

```

(Program.vb)
Imports System.IO
Imports System.Reflection
Imports DevExpress.XtraRichEdit
Imports DevExpress.XtraRichEdit.API.Native

Dim server As New RichEditDocumentServer()
server.LoadDocument("Texts\InlinePictures.rtf", DocumentFormat
Dim doc As Document = server.Document
' Insert an image from a file.
Dim rangeFound As DocumentRange = doc.FindAll("Visual Studio M
Dim pos As DocumentPosition = doc.Paragraphs(doc.Paragraphs.Ge
doc.Images.Insert(pos, DocumentImageSource.FromFile("Pictures\
' Insert an image from a stream.
pos = doc.Paragraphs(4).Range.Start
Dim imageToInsert As String = "information.png"
Dim a As System.Reflection.Assembly = System.Reflection.Assemb
Dim imageStream As Stream = a.GetManifestResourceStream("Resou
doc.Images.Insert(pos, DocumentImageSource.FromStream(imageStr
' Insert an image using its URI.
Dim imageUri As String = "http://i.gyazo.com/798a2ed48a3535c6c
Dim docHeader As SubDocument = doc.Sections(0).BeginUpdateHead
docHeader.Images.Append(DocumentImageSource.FromUri(imageUri,
doc.Sections(0).EndUpdateHeader(docHeader)
' Insert a barcode.
Dim barCode As New DevExpress.BarCodes.BarCode()
barCode.Symbology = DevExpress.BarCodes.Symbology.QRCode
barCode.CodeText = "http://www.devexpress.com"
barCode.CodeBinaryData = System.Text.Encoding.Default.GetBytes
barCode.Module = 0.5
Dim docFooter As SubDocument = doc.Sections(0).BeginUpdateFoot
docFooter.Images.Append(barCode.BarCodeImage)
doc.Sections(0).EndUpdateFooter(docFooter)

```

**Tip**

The `ReadOnlyDocumentImageCollection.Get` method is designed to get images within the specified Range. To call the method, use the **`RichEditDocumentServer.Document.Images.Get`** notation.

When a document is saved in RTF, inline pictures are saved twice by default: in the native format and as a metafile. Use the `RtfDocumentExporterCompatibilityOptions.DuplicateObjectAsMetafile` property to modify this behavior.

When a document is saved in HTML format, you can specify a location for storing images as external files, or process them before saving. To accomplish this, handle the `RichEditControl.BeforeExport` event and specify the `HtmlDocumentExporterOptions.UriExportType` and `IExporterOptions.TargetUri`. Moreover, you can implement your own class with the `IUriProvider` interface and specify it, instead of the default URI provider in the `SubDocument.GetHtmlText` method.

The `Document.HtmlText` method uses a special URI provider (the **`DataStringUriProvider`**), which converts an image to a base64-encoded representation

with the **"data:"** prefix. Therefore, if you obtain the HTML of a document via the `Document.HtmlText` method, all images are embedded in the page.

Hyperlinks and Bookmarks

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Hyperlinks and Bookmarks](#)

Hyperlinks are special fields of the [HYPERLINK](#) type. Hyperlinks can link you to external addresses or specific pages, or they can be used to point to bookmarks.

Bookmarks identify the location in a document or a document range.

Bookmark

A **Bookmark** exposes a document range with a name. The range can have a zero length, in which case the bookmark identifies the location within a document. The name of the Bookmark should start with a letter, and must be unique in the collection of bookmarks in the current document. This collection is accessible via the SubDocument.Bookmarks property.

To create a new bookmark, specify its name and add it to the document, use the BookmarkCollection.Create method. If the bookmark with the same name already exists, an **InvalidOperationException** exception is thrown with the message obtained from the **XtraRichEditStringId.Msg_DuplicateBookmark** resource string that reads "Bookmark with that name already exists in the document".

To select a bookmark, use the BookmarkCollection.Select method. The BookmarkCollection.Remove method deletes the specified bookmark.

Bookmarks in the document can be referenced by a **Hyperlink**. The Hyperlink.Anchor property specifies the bookmark to which the hyperlink is pointing.

Hyperlink

Hyperlinks are contained in the SubDocument.Hyperlinks collection of the document. To create a new hyperlink, use the HyperlinkCollection.Create method.

Headers and Footers

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Headers and Footers](#)

Each section in the document can have its own set of headers and footers. If no headers or footers are defined for a given section, the headers and footers from the previous section (if any) are used.

To get access to a header or footer, use the `Section.BeginUpdateHeader` - `Section.EndUpdateHeader` method pair or the `Section.BeginUpdateFooter` - `Section.EndUpdateFooter` method pair.

Note

The RichEditControl.Document property only provides access to the body of the main document. This means that you cannot access field or bookmark collections located in the header/footer using this property, only via the BeginUpdateHeader - EndUpdateHeader method pair.

The header (or footer) can be different for odd pages, even pages or the first page. Specify the `HeaderFooterType` in the `Section.BeginUpdateHeader` method.

Use the `Document.DifferentOddAndEvenPages` property to distinguish between odd and even pages in the document and the `Section.DifferentFirstPage` property to display a different header/footer (if any) for the first page of a section.

Tables

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Tables](#)

Table Support

The RichEditDocumentServer provides support for complex tables, as well as nested tables. An end-user can insert a table, edit the table layout by splitting and merging cells, and insert rows and columns. Borders and shading can be applied individually for each cell in the table.

You can use the Table API to add, delete or modify tables. The table in a document is represented by an object with the Table interface. To access rows and cells, the TableRow and TableCell interfaces are implemented. Tables are arranged in a TableCollection collection, accessible via the SubDocument.Tables property.

To access table content, use the **Range** properties of the cells, the rows and the table. Once you have obtained a document range, you can operate with it using its own methods, or via the Document or SubDocument methods. See the [Characters](#) and [Paragraphs](#) topics for more information on formatting.

Table API - Useful Members

Member	Description
SubDocument.Tables	Provides access to a collection of tables in the document.
TableCollection.Add	Creates a new table.
Table.BeginUpdate	Starts modifying a table.
Table.EndUpdate	Finishes modifying a table.
Table.Rows	Provides access to a collection of rows in the table.
Table.Cell	Provides access to a cell in a table.
TableRow.Cells	Provides access to a collection of cells in a row.
TableCell.Range	Gets the document range occupied by a cell. You can use the SubDocument.BeginUpdateCharacters method to change character formatting. Use the SubDocument.GetParagraphs method to access paragraphs within a cell.
Table.ForEachRow	Specifies a delegate executed for each row in a table.
Table.ForEachCell	Specifies the delegate executed for each cell in a table.
Document.TableStyles	Provides access to a collection of styles applied to tables.
Document.DefaultTableProperties	Specifies the default formatting for document tables.

Sections

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Sections](#)

A document is divided into logical Document.Sections. Sections are contained within a collection represented by a SectionCollection object. Each section defines its own page settings such as paper size (the SectionPage.PaperKind property), page orientation (the SectionPage.Landscape property) and margins. The Section.Page and Section.Margins properties enable you to access these settings.

You can specify different settings for the first page of a section using the Section.DifferentFirstPage property.

The [Headers and Footers](#) in the document belong to a specific section. To edit header or footer content, use the Section.BeginUpdateHeader - Section.EndUpdateHeader method pair and the Section.BeginUpdateFooter - Section.EndUpdateFooter method pair, respectively.

The Line numbering feature can be activated for a particular section via the Section.LineNumbering property.

To insert a section in an arbitrary position, use the Document.InsertSection method.

Styles

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Styles](#)

A document model contains collections of **styles**. There are three types of styles - a **paragraph style**, a **character style** and a **table style**. Styles are accessible via the Document.ParagraphStyles, Document.CharacterStyles and Document.TableStyles properties, respectively.

Use the CharacterStyleCollection.CreateNew, ParagraphStyleCollection.CreateNew and TableStyleCollection.CreateNew methods to create new styles. After creating a new style, specify style formatting and add it to the corresponding style collection using the collection's **Add** method. You do not have to specify all formatting properties. Instead, you can specify the parent style (the CharacterStyle.Parent, ParagraphStyle.Parent and TableStyle.Parent properties), and modify only the required characteristics.

Some styles can act as a paragraph style or a character style, depending on the range to which it is applied. Such styles are known as **linked** styles. If a linked style is applied to a selection that does not include an entire paragraph, the selected text is formatted with the character formatting settings of the linked style. If a selection includes an entire paragraph, both character and paragraph formatting settings are applied. If a selection is only a caret position that contains no characters, the linked style behaves like a paragraph style.

You can delete a style from the collection of styles in the document using the CharacterStyleCollection.Delete, ParagraphStyleCollection.Delete or TableStyleCollection.Delete methods. A deleted style can no longer be applied to text or saved in a document file.

Styles are inherent to the document, so you cannot add a style contained in the style collection of one document to the style collection of another document. In this case, create a new style and copy its properties, one by one. When performing mail merge, use the MailMergeOptions.CopyTemplateStyles property to copy template styles to the merged document.

Numbered and Bulleted Lists

[Office File API](#) > [Word Processing Document API](#) > [WordProcessing Document](#) > [Document Elements](#) > [Numbered and Bulleted Lists](#)

RichEditDocumentServer supports bulleted, numbered and mixed lists.

Numbered and bulleted lists are represented by specific collections of paragraphs. Each paragraph in a document may contain numbering information. Numbering information for a specific paragraph is determined by its Paragraph.ListIndex and Paragraph.ListLevel properties.

The Paragraph.ListIndex property value is the index in the document's NumberingListCollection collection. This collection is available via the Document.NumberingLists property.

The **NumberingListCollection** is composed of items with a NumberingList interface. Each item is an instance of a certain abstract numbering definition, which is represented by the AbstractNumberingList interface. A collection of abstract numbering definitions is accessible via the Document.AbstractNumberingLists property.

The Paragraph.ListLevel property of a paragraph indicates its level in the list. To specify formatting for a numbered paragraph, modify the properties of the corresponding level. Level properties are contained in an object that has the ListLevelProperties interface, accessible via the NumberingList.Levels property. This change will apply to all paragraphs with a certain level linked to this numbered list. If a property is not specified for the numbered list, the definition for the corresponding level comes from the abstract numbering definition (AbstractNumberingList) linked to this numbered list.

Bulleted lists are numbered lists with the NumberingListBase.NumberingType property set to NumberingType.Bullet and the ListLevel.BulletLevel property set to **true**.

Note

To remove numbering for a particular paragraph, set its ListIndex to -1.

Merge and Split Documents

[Office File API](#) > [Word Processing Document API](#) > [Merge and Split Documents](#)

This section contains the following topics:

- [Merge Documents](#)

Merge Documents

[Office File API](#) > [Word Processing Document API](#) > [Merge and Split Documents](#) > [Merge Documents](#)

This topic describes how to use the RichEditDocumentServer to merge documents and [keep document formatting](#).

Merge Documents

1. Load documents to be merged in separate RichEditDocumentServer instances.
2. Successively insert the content of each server in the main document using one of the following members:
 - SubDocument.AppendDocumentContent - Appends content from the file.
 - SubDocument.InsertDocumentContent - Inserts content from the file in the current document at the specified position.
3. Call the IRichEditDocumentServer.SaveDocument method to save the resulting document.

C#

```
var richEditDocumentServer1 = new RichEditDocumentServer();
richEditDocumentServer1.LoadDocument("document1.docx", DocumentFormat.OpenXml);
var richEditDocumentServer2 = new RichEditDocumentServer();
richEditDocumentServer2.LoadDocument("document2.docx", DocumentFormat.OpenXml);
richEditDocumentServer1.Document.AppendDocumentContent(richEditDocumentServer2.Document.Range);
richEditDocumentServer1.SaveDocument("result.docx", DocumentFormat.OpenXml);
```

Visual Basic

```
Dim richEditDocumentServer1 = New RichEditDocumentServer()
richEditDocumentServer1.LoadDocument("document1.docx", DocumentFormat.OpenXml)
Dim richEditDocumentServer2 = New RichEditDocumentServer()
richEditDocumentServer2.LoadDocument("document2.docx", DocumentFormat.OpenXml)
richEditDocumentServer1.Document.AppendDocumentContent(richEditDocumentServer2.Document.Range)
richEditDocumentServer1.SaveDocument("result.docx", DocumentFormat.OpenXml)
```

Merge Documents into a PDF File

Call the RichEditDocumentServer.ExportToPdf method to export the result to a PDF file. You can also export all documents to separate PDF files and merge them using the [PDF Document API](#). Refer to the [How to: Merge Documents into a Single PDF File](#) topic for more information.

Keep Document Formatting

Direct Formatting and Styles

Pass one of InsertOptions enumeration members to SubDocument.AppendDocumentContent or SubDocument.InsertDocumentContent method as an insertOptions parameter to manage how the formatting is applied to the inserted content. The table below shows how formatting is applied depending on the selected value:

Enumeration Value	Direct Formatting	Styles (character, paragraph, table)	Non-textual objects (tables, floating objects, comments, etc.)	Tabs
InsertOptions.KeepSourceFormatting	The copied text retains its formatting.	New styles (styles that don't exist in the target document) are copied. If character or paragraph style conflict occurs, all inserted style's properties are applied as direct formatting, but the style itself is not transferred. Destination's styles with default settings are substituted with	Non-textual elements are retained.	All page properties are retained.

		inserted styles of the same name. Conflicted table and numbering list styles are copied to the document. The style name is a combination of conflicted styles' names (e.g., <i>Normal1</i> and <i>Normal1</i> are combined into <i>Normal11</i>).		
InsertOptions.KeepTextOnly	The inserted text loses its formatting and takes on direct formatting applied to the target text range. Numbering lists are converted to simple text.	All inserted styles are not retained.	Floating objects, shapes and comments are discarded. Tables are converted into a series of paragraphs.	Tabs are lost.
InsertOptions.MergeFormatting	<p>The inserted text retains the following character properties:</p> <ul style="list-style-type: none"> • CharacterPropertiesBase.Bold • CharacterPropertiesBase.Italic • CharacterPropertiesBase.Hidden • CharacterPropertiesBase.Language • CharacterPropertiesBase.NoProof • CharacterPropertiesBase.Subscript • CharacterPropertiesBase.Superscript • CharacterPropertiesBase.Underline <p>Other character properties and all paragraph properties are ignored.</p>	Inserted style's character properties are applied as direct, but the style itself is not copied to the target document. Style paragraph properties are ignored. Table styles are not retained. Numbering list styles are transferred to the target document.	Non-textual elements are retained.	Tabs are ignored.
InsertOptions.UseDestinationStyles	The copied text keeps its format options.	New styles (styles that don't exist in the target document) are copied. If style conflict occurs, the inserted content's styles are ignored. Destination's styles with default settings are substituted with inserted styles of the same name.	Non-textual elements are retained.	All page properties are retained.

 **Note**

The RichEditDocumentServer does not currently support themes. When merging documents with themes, the result may lose theme-related formatting.

Headers, Footers and Page Settings

New content is appended to the last section of the original document, and all information regarding the former section (header, footer, page layout, etc.) is lost. Perform the following steps to keep the inserted content's page settings:

- Create a new section by calling the `Document.AppendSection` or `Document.InsertSection` method.
- Call the `Section.UnlinkFooterFromPrevious` or `Section.UnlinkHeaderFromPrevious` to unlink the target and the source document's header or footer from each other.
- Copy the required `SectionPage` options to retain the original document's page settings in the in the merged document.
- Start editing both documents' headers or footers by calling the `Section.BeginUpdateHeader` or `Section.BeginUpdateFooter` method.
- Clear the target document's header by calling the `SubDocument.Delete` method. Then insert the source document's header or footer content using the `SubDocument.InsertDocumentContent` method.
- Finalize editing by calling the `Section.EndUpdateHeader` or `Section.EndUpdateFooter` method for both documents.

See Also

A full code example is available here: [How to merge documents with headers and footers into a single document](#)

C#

```
targetDoc.AppendSection();
int targetSectionIndex = targetDoc.Sections.Count - 1;
int sourceSectionIndex = sourceDoc.Sections.Count - 1;
Section targetSection = targetDoc.Sections[targetSectionIndex];
Section sourceSection = sourceDoc.Document.Sections[sourceSectionIndex];
CopySectionSettings(targetSection, sourceSection);
targetSection.UnlinkHeaderFromPrevious();
targetSection.UnlinkFooterFromPrevious();
AppendHeader(sourceSection, targetSection, HeaderFooterType.Odd);
private static void AppendHeader(Section sourceSection, Section targetSection, HeaderFooterType headerFoot
{
    //Initialize the update session
    SubDocument source = sourceSection.BeginUpdateHeader(headerFooterType);
    SubDocument target = targetSection.BeginUpdateHeader(headerFooterType);
    //Clear the target header and insert the source header's content in it
    target.Delete(target.Range);
    target.InsertDocumentContent(target.Range.Start, source.Range, InsertOptions.KeepSourceFormatting);
    // Delete empty paragraphs
    DocumentRange emptyParagraph = target.CreateRange(target.Range.End.ToInt() - 2, 2);
    target.Delete(emptyParagraph);
//Finalize the update
    sourceSection.EndUpdateHeader(source);
    targetSection.EndUpdateHeader(target);
}
//Copy page settings from one section to another
private static void CopySectionSettings(Section section, Section sourceSection)
{
    section.Page.Landscape = sourceSection.Page.Landscape;
    section.Page.PaperKind = sourceSection.Page.PaperKind;
}
```

Visual Basic

```
targetDoc.AppendSection()  
Private targetSectionIndex As Integer = targetDoc.Sections.Count - 1  
Private sourceSectionIndex As Integer = sourceDoc.Sections.Count - 1  
Private targetSection As Section = targetDoc.Sections(targetSectionIndex)  
Private sourceSection As Section = server2.Document.Sections(sourceSectionIndex)  
targetSection.UnlinkHeaderFromPrevious()  
targetSection.UnlinkFooterFromPrevious()  
AppendHeader(sourceSection, targetSection, HeaderFooterType.Odd)  
private static void AppendHeader(Section sourceSection, Section targetSection, HeaderFooterType headerFoot  
    'Initialize the update session  
    Dim source As SubDocument = sourceSection.BeginUpdateHeader(headerFooterType)  
    Dim target As SubDocument = targetSection.BeginUpdateHeader(headerFooterType)  
    'Clear the target header and insert the source header's content to it  
    target.Delete(target.Range)  
    target.InsertDocumentContent(target.Range.Start, source.Range, InsertOptions.KeepSourceFormatting)  
    'Delete empty paragraphs  
    Dim emptyParagraph As DocumentRange = target.CreateRange(target.Range.End.ToInt() - 2, 2)  
    target.Delete(emptyParagraph)  
    'Finalize the update  
    sourceSection.EndUpdateHeader(source)  
    targetSection.EndUpdateHeader(target)  
    'Copy page settings from one section to another  
Private Shared Sub CopySectionSettings(ByVal section As Section, ByVal sourceSection As Section)  
    section.Page.Landscape = sourceSection.Page.Landscape  
    section.Page.Landscape = sourceSection.Page.Landscape  
End Sub
```

Import and Export

[Office File API](#) > [Word Processing Document API](#) > [Import and Export](#)

- [Supported Formats](#)
- [Load Documents](#)
- [Save Documents](#)
- [Get or Set the Document Content](#)

Supported Formats

The RichEditDocumentServer supports the following document formats:


- Plain Text;
- RTF (Rich Text Format);
- DOCX (MS Office Word 2007 -... format);
- DOC (Microsoft Word 97-2003 format);
- WordML (MS Office Word 2003 XML format);
- OpenDocument (implemented by the OpenOffice.org office suite);
- HTML;
- MHTML (web page archive format);
- EPUB (Electronic Publication);
- PDF format (export only)

Load Documents

The RichEditDocumentServer provides the following methods to import the document and specify its options:

Member	Description
RichEditDocumentServer.LoadDocument	Loads a document from a file or stream. The document format can be specified manually or detected automatically.
RichEditDocumentServer.LoadDocumentTemplate	Loads a document from the specified file or stream so that it cannot be overwritten automatically.
Document.LoadDocument	Loads a document from the specified file or stream. Its content determines the file format or specified manually.
SubDocument.InsertDocumentContent	Inserts content from the specified range into the current document at the specified position.
SubDocument.AppendDocumentContent	Appends content from the specified range to the end of the current document.

Handle the RichEditDocumentServer.BeforeImport event to specify the format-specific import options (accessed by the RichEditControlOptionsBase.Import property), as shown in the code snippet below.

 Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T418535 .	

C#	
----	--

```
(ImportActions.cs)
server.LoadDocument("Documents\\TerribleRevengeKOI8R.txt");
server.BeforeImport += BeforeImportHelper.BeforeImport;
class BeforeImportHelper
{
    public static void BeforeImport(object sender, BeforeImportEventArgs e)
    {
        if (e.DocumentFormat == DocumentFormat.PlainText)
        {
            ((PlainTextDocumentImporterOptions)e.Options).Encoding = Encoding.UTF32;
        }
    }
}
```

Visual Basic

```
(ImportActions.vb)
server.LoadDocument("Documents\TerribleRevengeKOI8R.txt")
AddHandler server.BeforeImport, AddressOf BeforeImportHelper.BeforeImport
Private Class BeforeImportHelper
    Public Shared Sub BeforeImport(ByVal sender As Object, ByVal e As BeforeImportEventArgs)
        If e.DocumentFormat = DocumentFormat.PlainText Then
            CType(e.Options, PlainTextDocumentImporterOptions).Encoding = Encoding.UTF32
        End If
    End Sub
End Class
```

The table below lists the document formats the RichEditDocumentServer supports, and the API used to specify the necessary import options:

Format	Import Options
Plain Text	RichEditDocumentImportOptions.PlainText
Rich Text Format	RichEditDocumentImportOptions.Rtf
DOCX	RichEditDocumentImportOptions.OpenXml
DOC	RichEditDocumentImportOptions.Doc
WordML	RichEditDocumentImportOptions.WordML
MHT	RichEditDocumentImportOptions.Mht

Use the RichEditControlOptionsBase.DocumentCapabilities to access the DocumentCapabilitiesOptions instance and restrict importing certain document elements (bookmarks, comments, fields, etc.).

Note

Hyperlink fields represent hyperlinks in the RichEditDocumentServer, therefore the DocumentCapabilitiesOptions.Fields property affects the hyperlink loading process when importing the documents. Set the RichEditControlCompatibility.LoadHyperlinkAsField property to **false** before calling the **InitializeComponent** method to allow loading hyperlinks when loading fields is disabled.

You can use the Document.DefaultCharacterProperties or Document.DefaultParagraphProperties property to set every imported document's default character or paragraph formatting options.

HTML Documents

Loaded HTML documents are parsed and transformed into the internal document model. Not every HTML tag can be converted into a corresponding document model element. Refer to the [HTML Tag Support](#) topic for a full list of supported tags.

When exporting a document to HTML, the document model is parsed and transformed again, resulting in a document that is different from the original HTML document.

You can implement your IUriStreamProvider or IUriStreamService descendants and override the IUriStreamProvider.GetStream method to manage decoding external contents in an HTML file when an HTML document is loaded. When the RichEditControl needs to load data from a specific URI, your method is executed instead of the default one.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4164>.

This example demonstrates how to implement the `IUriStreamProvider` interface to create a custom data stream provider for the `IUriProviderService` service.

C#

```
(ImageStreamProvider.cs)
public class ImageStreamProvider : IUriStreamProvider {
    static readonly string prefix = "dbimg://";
    DataTable table;
    string columnName;
    public ImageStreamProvider(DataTable sourceTable, string imageColumn)
    {
        this.table = sourceTable;
        this.columnName = imageColumn;
    }
    public Stream GetStream(string uri) {
        uri = uri.Trim();
        if (!uri.StartsWith(prefix))
            return null;
        string strId = uri.Substring(prefix.Length).Trim();
        int id;
        if (!int.TryParse(strId, out id))
            return null;
        DataRow row = table.Rows.Find(id);
        if (row == null)
            return null;
        byte[] bytes = row[columnName] as byte[];
        if (bytes == null)
            return null;
        // Use this approach to trim the OLE header off the image
        // See also: http://www.devexpress.com/issue=Q233460,
        // http://social.msdn.microsoft.com/Forums/en-US/sqldataaccess/thr
        MemoryStream memoryStream = new MemoryStream();
        int oleHeaderOffset = 78;
        memoryStream.Write(bytes, oleHeaderOffset, bytes.Length - oleHeaderOffset);
        return memoryStream;
    }
}
```

Visual Basic

```

(ImageStreamProvider.vb)
Public Class ImageStreamProvider
    Implements IUriStreamProvider
    Private Shared ReadOnly prefix As String = "dbimg://"
    Private table As DataTable
    Private columnName As String
    Public Sub New(ByVal sourceTable As DataTable, ByVal imageColumn As St
        Me.table = sourceTable
        Me.columnName = imageColumn
    End Sub
    Public Function GetStream(ByVal uri As String) As Stream Implements IU
        uri = uri.Trim()
        If (Not uri.StartsWith(prefix)) Then
            Return Nothing
        End If
        Dim strId As String = uri.Substring(prefix.Length).Trim()
        Dim id As Integer
        If (Not Integer.TryParse(strId, id)) Then
            Return Nothing
        End If
        Dim row As DataRow = table.Rows.Find(id)
        If row Is Nothing Then
            Return Nothing
        End If
        Dim bytes() As Byte = TryCast(row(columnName), Byte())
        If bytes Is Nothing Then
            Return Nothing
        End If
        ' Use this approach to trim the OLE header off the image
        ' See also: http://www.devexpress.com/issue=Q233460,
        ' http://social.msdn.microsoft.com/Forums/en-US/sqldataaccess/thre
        Dim memoryStream As New MemoryStream()
        Dim oleHeaderOffset As Integer = 78
        memoryStream.Write(bytes, oleHeaderOffset, bytes.Length - oleHeade
        Return memoryStream
    End Function
End Class

```

Save Documents

Use the following methods to save a document manually:

- RichEditDocumentServer.SaveDocument - Saves the control's document to a file, specifying the document's format.
- Document.SaveDocument - Saves the document to a file, specifying the document's format.

Handle the RichEditDocumentServer.BeforeExport event to specify the format-specific export options (accessed by the RichEditControlOptionsBase.Export property), as shown in the code snippet:

🔗 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#
<pre>(ExportActions.cs) server.LoadDocument("Documents\\Grimm.docx"); server.BeforeExport += BeforeExportHelper.BeforeExport; server.SaveDocument("Document_HTML.html", DocumentFormat.Html); System.Diagnostics.Process.Start("Document_HTML.html"); class BeforeExportHelper { public static void BeforeExport(object sender, BeforeExportEventArgs e) { DevExpress.XtraRichEdit.Export.HtmlDocumentExporterOptions options = e.Options; if (options != null) { options.CssPropertiesExportType = DevExpress.XtraRichEdit.Export.CssPropertiesExportType.Html; options.HtmlNumberingListExportFormat = DevExpress.XtraRichEdit.Export.HtmlNumberingListExportFormat.Default; options.TargetUri = "Document_HTML.html"; } } }</pre>

Visual Basic
<pre>(ExportActions.vb) server.LoadDocument("Documents\\Grimm.docx") AddHandler server.BeforeExport, AddressOf BeforeExportHelper.BeforeExport server.SaveDocument("Document_HTML.html", DocumentFormat.Html) System.Diagnostics.Process.Start("Document_HTML.html") Private Class BeforeExportHelper Public Shared Sub BeforeExport(ByVal sender As Object, ByVal e As BeforeExportEventArgs) Dim options As DevExpress.XtraRichEdit.Export.HtmlDocumentExporterOptions = e.Options If options IsNot Nothing Then options.CssPropertiesExportType = DevExpress.XtraRichEdit.Export.CssPropertiesExportType.Html options.HtmlNumberingListExportFormat = DevExpress.XtraRichEdit.Export.HtmlNumberingListExportFormat.Default options.TargetUri = "Document_HTML.html" End If End Sub End Class</pre>

Use the API from the table below to set the corresponding export options:

Format	Export Options
Plain Text	RichEditDocumentExportOptions.PlainText
Rich Text Format	RichEditDocumentExportOptions.Rtf
MS Word 2007 (OpenXML)	RichEditDocumentExportOptions.OpenXml
DOC	RichEditDocumentExportOptions.Doc

WordML	RichEditDocumentExportOptions.WordML
HTML	RichEditDocumentExportOptions.Html
MHT	RichEditDocumentExportOptions.Mht
PDF	PdfExportOptions

Use the RichEditControlOptionsBase.DocumentSaveOptions property to define the saved document's name and format.

HTML Documents

When exporting a document to HTML format, you can specify a location for storing images as external files or process them before saving. To do this, handle the RichEditDocumentServer.BeforeExport event and specify the HtmlDocumentExporterOptions.UriExportType and IExporterOptions.TargetUri. Moreover, you can implement your class with the IUriProvider interface and override the IUriProvider.CreateImageUri and IUriProvider.CreateCssUri methods. Then, it can be used instead of the default URI provider when the SubDocument.GetHtmlText method is called.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E1604>

.

C#	
----	--

```
(Form1.cs)
public class CustomUriProvider : DevExpress.Office.Services.IUriProvider
{
    string rootDirectory;
    public CustomUriProvider(string rootDirectory) {
        if (String.IsNullOrEmpty(rootDirectory))
            DevExpress.Office.Utils.Exceptions.ThrowArgumentException("rootDirectory");
        this.rootDirectory = rootDirectory;
    }
    public string CreateCssUri(string rootUri, string styleText, string rootUri)
    {
        string cssDir = String.Format("{0}\\{1}", this.rootDirectory, rootUri);
        if (!Directory.Exists(cssDir))
            Directory.CreateDirectory(cssDir);
        string cssFileName = String.Format("{0}\\style.css", cssDir);
        File.AppendAllText(cssFileName, styleText);
        return GetRelativePath(cssFileName);
    }
    public string CreateImageUri(string rootUri, DevExpress.Office.Utils.IImage image)
    {
        string imagesDir = String.Format("{0}\\{1}", this.rootDirectory, rootUri);
        if (!Directory.Exists(imagesDir))
            Directory.CreateDirectory(imagesDir);
        string imageName = String.Format("{0}\\{1}.png", imagesDir, Guid.NewGuid().ToString());
        image.NativeImage.Save(imageName, ImageFormat.Png);
        return GetRelativePath(imageName);
    }
    string GetRelativePath(string path) {
        string substring = path.Substring(this.rootDirectory.Length);
        return substring.Replace("\\", "/").Trim('/');
    }
}
```

Visual Basic

```

(Form1.vb)
Public Class CustomUriProvider
    Implements DevExpress.Office.Services.IUriProvider
    Private rootDirectory As String
    Public Sub New(ByVal rootDirectory As String)
        If String.IsNullOrEmpty(rootDirectory) Then
            DevExpress.Office.Utils.Exceptions.ThrowArgumentException("rootDirectory")
        End If
        Me.rootDirectory = rootDirectory
    End Sub
    Public Function CreateCssUri(ByVal rootUri As String, ByVal styleText As String) As String
        Dim cssDir As String = String.Format("{0}\{1}", Me.rootDirectory, rootUri)
        If Not Directory.Exists(cssDir) Then
            Directory.CreateDirectory(cssDir)
        End If
        Dim cssFileName As String = String.Format("{0}\style.css", cssDir)
        File.AppendAllText(cssFileName, styleText)
        Return GetRelativePath(cssFileName)
    End Function
    Public Function CreateImageUri(ByVal rootUri As String, ByVal image As Image) As String
        Dim imagesDir As String = String.Format("{0}\{1}", Me.rootDirectory, rootUri)
        If Not Directory.Exists(imagesDir) Then
            Directory.CreateDirectory(imagesDir)
        End If
        Dim imageName As String = String.Format("{0}\{1}.png", imagesDir, image.Name)
        image.Save(imageName, ImageFormat.Png)
        Return GetRelativePath(imageName)
    End Function
    Private Function GetRelativePath(ByVal path As String) As String
        Dim substring As String = path.Substring(Me.rootDirectory.Length)
        Return substring.Replace("\", "/").Trim("/")
    End Function
End Class

```

Use the `HtmlDocumentExporterOptions.EmbedImages` option to store images in base-64 encoding. The `HtmlDocumentExporterOptions.TabMarker` property allows you to substitute **Tab (0x09) characters** with any symbol combination in HTML output.

Specify the corresponding paragraph's `Paragraph.OutlineLevel` property to mark the text in HTML output with **H1 - H6** tags. Paragraphs with outline levels higher than 6 are exported as text enclosed in a **P** tag.

RTF Documents

When saving a document in an RTF format, inline pictures can be stored twice - in the native format and as a Windows Metafile. If an editing application is unable to process the native picture format, it can display a Windows Metafile. Only pictures in the native format are saved by default to reduce the file size. Use the `RtfDocumentExporterCompatibilityOptions.DuplicateObjectAsMetafile` property to modify this behavior.

All document styles are saved by default, which could increase the saved document's size. You can delete unused styles from the `Document.CharacterStyles`,

Document.ParagraphStyles and Document.TableStyles style collections before exporting to reduce the size of data in RTF format. Set the RtfDocumentExporterOptions.ExportTheme property to **false** to exclude the color theme information from the file.

PDF Documents

Call the RichEditDocumentServer.ExportToPdf method to export documents to PDF format. Use the PdfExportOptions class properties to specify advanced export settings, as demonstrated below.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ExportActions.cs)
server.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml)
//Specify export options:
PdfExportOptions options = new PdfExportOptions();
options.DocumentOptions.Author = "Mark Jones";
options.Compressed = false;
options.ImageQuality = PdfJpegImageQuality.Highest;
//Export the document to the stream:
using (FileStream pdfFileStream = new FileStream("Document_PDF.pdf", FileMode.Create))
{
    server.ExportToPdf(pdfFileStream, options);
}
System.Diagnostics.Process.Start("Document PDF.pdf");
```

Visual Basic

```
(ExportActions.vb)
server.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml)
'Specify export options:
Dim options As New PdfExportOptions()
options.DocumentOptions.Author = "Mark Jones"
options.Compressed = False
options.ImageQuality = PdfJpegImageQuality.Highest
'Export the document to the stream:
Using pdfFileStream As New FileStream("Document_PDF.pdf", FileMode.Create)
    server.ExportToPdf(pdfFileStream, options)
End Using
System.Diagnostics.Process.Start("Document PDF.pdf")
```

Get or Set Document Content

Use one of the following properties to get or set the document content in a specific format:

- Plain Text - Document.Text;
- Rich Text Format - Document.RtfText;

- OpenXML - Document.OpenXmlBytes;
- WordML - Document.WordMLText;
- HTML - Document.HtmlText;
- MHT - Document.MhtText.

Tip

Use the SubDocument's **Get...** methods to retrieve and format a part of the document. Refer to the [Positions and Ranges](#) topic for details.

See Also

[Supported Formats](#)

[Export in Examples section](#)

[Export to PDF](#)

Fields

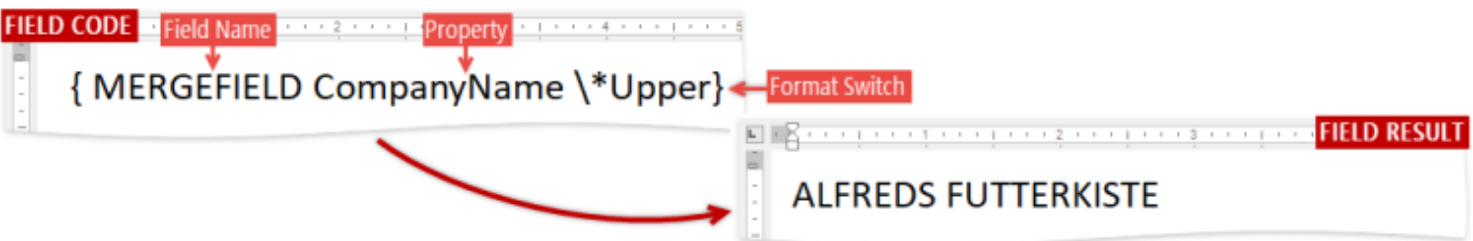
[Office File API](#) > [Word Processing Document API](#) > [Fields](#)

This topic consists the following sections:

- [Overview](#)
- [Insert and Modify Fields](#)
- [Update Fields](#)
- [Field Options](#)

Overview

The Field is a set of codes that instructs the RichEditDocumentServer to insert text or graphics into a document automatically. The field code syntax is illustrated and described below.



- Field Name** - the name of the current field. Refer to the [Field Codes](#) section for a list of all available fields.
- Property** - instruction/variable used in a particular field. This is an optional element.
- Switch** - an additional parameter that offers more information. This is an optional element. Refer to the [Format Switches](#) section for a list of supported format switches.

A field in the document consists of two ranges - the Field.CodeRange and the Field.ResultRange. Use the Field.Range property to obtain the total range the field occupies.

The RichEditDocumentServer supports the following field types:

- 1. Non-MailMerge**
Regular fields used to insert a simple data, such as [DATE](#), [TIME](#) or [PAGE](#). The Non-MailMerge field results are shown after the update.
- 2. MailMerge**
Fields used in the [Mail Merge](#) process. They get a value only if there is a mail merge data source bound to the RichEditDocumentServer. In the resulting document, MailMerge fields are evaluated and substituted with the resulting values.
- 3. Mixed**
Fields which do not require a data source to get its value, such as [INCLUDEPICTURE](#) and [CREATEDATE](#). They are substituted with the resulting values during mail merge.

Insert and Modify Fields

Members from the table below allow you to insert and modify the field programmatically.

Member	Description
SubDocument.BeginUpdate	Opens the document for editing.
FieldCollection.Create	Creates a new Field object.
SubDocument.Fields	Provides access to the collection of fields in the current document.
Field.Range	Gets the document range occupied by the field.
Field.CodeRange	Gets the range containing the field codes.
Field.ResultRange	Gets the range containing the field result.

SubDocument.InsertText	Inserts the specified text at the specified position.
SubDocument.EndUpdate	Finalizes the modification.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(FieldActions.cs)
Document document = server.Document;
document.BeginUpdate();
document.Fields.Create(document.CaretPosition, "DATE");
document.EndUpdate();
for (int i = 0; i < document.Fields.Count; i++)
{
    string fieldCode = document.GetText(document.Fields[i].CodeRange);
    if (fieldCode == "DATE")
    {
        DocumentPosition position = document.Fields[i].CodeRange.End;
        document.InsertText(position, @" "M / d / yyyy HH: mm:ss");
    }
}
document.Fields.Update();
```

Visual Basic

```
(FieldActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
document.Fields.Create(document.CaretPosition, "DATE")
document.EndUpdate()
For i As Integer = 0 To document.Fields.Count - 1
    Dim fieldCode As String = document.GetText(document.Fields(i).CodeRange)
    If fieldCode = "DATE" Then
        Dim position As DocumentPosition = document.Fields(i).CodeRange.End
        document.InsertText(position, " \@ "M / d / yyyy HH: mm:ss")
    End If
Next i
document.Fields.Update()
```

Tip

The header and footer in the RichEditCotnrol document do not belong to the main document body. Use the Section.BeginUpdateHeader - Section.EndUpdateHeader or Section.BeginUpdateFooter - Section.EndUpdateFooter paired methods to obtain the header or footer. Refer to the [How to: Create and Modify Header](#) topic for details.

Update Fields

Most fields are updated automatically when the document is saved or printed, or during the mail merge operation. Use the Field.Update method to update the field on demand. Specify the DocumentImporterOptions.UpdateField property to set what document fields should be updated automatically when loading the document. Use the **docServer.Options.Import.DocumentFormat.UpdateField** notation (where DocumentFormat is one of the RichTextEditDocumentImportOptions properties) to access this property.

Set the field's Field.Locked property to **true** to prevent a specific field from being updated. Use the FieldOptions.UpdateLockedFields option to allow updating locked fields.

The [DOCVARIABLE](#) field is not updated during mail merge or document insertion. Handle the RichEditDocumentServer.CalculateDocumentVariable event to update the DOCVARIABLE fields' values.

The RichEditDocumentServer.CalculateDocumentVariable event does not occur for a locked DOCVARIABLE field, and the field value remains unchanged. Set the FieldOptions.UpdateLockedFields option to the UpdateLockedFields.DocVariableOnly value to update [DOCVARIABLE](#) fields.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E3099>

.

Tip

You can transform a mail merge template to a self-contained document in the same RichEditDocumentServer instance by using current data. Refer to the [How to replace document fields with their values](#) example for details.

Field Options

Use the Field.ShowCodes property to change the target field's display mode. You can also use ShowAllFieldCodesCommand or ShowAllFieldResultsCommand commands to change the display mode of all document fields.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

This code snippet displays field codes for all fields in the main document body. The fields containing in the header or footer belong to a different FieldCollection. Use the Section.BeginUpdateHeader- Section.EndUpdateHeader or Section.BeginUpdateFooter - Section.EndUpdateFooter paired methods to retrieve the header or footer document.

C#

```
(FieldActions.cs)
Document document = server.Document;
document.LoadDocument("MailMergeSimple.docx", DocumentFormat.OpenXml);
for (int i = 0; i < document.Fields.Count; i++)
{
    document.Fields[i].ShowCodes = true;
}
```

Visual Basic

```
(FieldActions.vb)
Dim document As Document = server.Document
document.LoadDocument("MailMergeSimple.docx", DocumentFormat.OpenXml)
For i As Integer = 0 To document.Fields.Count - 1
    document.Fields(i).ShowCodes = True
Next i
```

Use the RichEditControlOptionsBase.Fields to specify how RichEditDocumentServer should process and display document fields, as shown in the code snippet below:

C#

```
FieldOptions fieldOptions = docServer.Options.Fields;  
fieldOptions.HighlightMode = FieldsHighlightMode.Auto;  
fieldOptions.HighlightColor = System.Drawing.Color.LightSalmon;  
fieldOptions.UseCurrentCultureDateTimeFormat = true;  
fieldOptions.ThrowExceptionOnInvalidFormatSwitch = true;  
fieldOptions.UpdateFieldsInTextBoxes = true;
```

Visual Basic

```
Dim fieldOptions As FieldOptions = docServer.Options.Fields  
fieldOptions.HighlightMode = FieldsHighlightMode.Auto  
fieldOptions.HighlightColor = System.Drawing.Color.LightSalmon  
fieldOptions.UseCurrentCultureDateTimeFormat = True  
fieldOptions.ThrowExceptionOnInvalidFormatSwitch = True  
fieldOptions.UpdateFieldsInTextBoxes = True
```

Set the RichEditMailMergeOptions.ViewMergedData to **true** or execute the ToggleViewMergedDataCommand command to display the MERGEFIELD fields code results.

Set the DocumentCapabilitiesOptions.Fields property to DocumentCapability.Hidden or DocumentCapability.Disabled to restrict inserting fields to the document.

Field Codes

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#)

Document fields are used as placeholders for data that might change in a document, as well as for mail merge operations. For more information on fields, see the [Fields](#) document.

There are three types of fields - **MailMerge**, **Non-MailMerge** and **Mixed**. MailMerge fields are used in the mail merge process, and get a value only if there is a merge data source bound to the RichEditControl. Otherwise, MailMerge fields are displayed as field placeholders that show the name of a corresponding data column, encompassed in angular brackets as follows:

<<Company Name>> for the MERGEFIELD coded as { MERGEFIELD CompanyName }

A significant difference between **MailMerge** and **Non-MailMerge** fields is in their behavior when the Document.MailMerge (or the IRichEditDocumentServer.MailMerge) method is executed. The resulting document does not contain fields of the MailMerge type. MailMerge fields are evaluated and substituted with the resulting values.

Results for Non-MailMerge fields are displayed when the field is updated.

Mixed field types denote fields that do not require a data source for evaluation, but during mail merge they are substituted with their values. CREATEDATE and INCLUDEPICTURE are examples of such fields.

Syntax

{ FIELDNAME Properties Optional_Switches }

- **FIELDNAME** - This is the name of the field.
- **Properties** - These are any instructions or variables that are used in a particular field. Not all fields have parameters, and in some fields, parameters are optional.
- **Optional switches** - These are any optional settings that are available for a particular field. Not all fields have switches available, other than those that control the formatting of the field results.

For example, the {MERGEFIELD Weather.condition *Upper} field denotes the [MERGEFIELD](#) field bound to the Weather.condition data field, which displays the text in all caps.

Supported Fields

Field Name	Functionality
AUTHOR	Inserts the name of a person or organization who created the document.
COMMENTS	Inserts the document notes.
CREATEDATE	Inserts the current date and time. After a mail merge the field is replaced with the date and time of the mail merge operation.
DATE	Inserts the current date and time.
DOCPROPERTY	Inserts the value of the document property specified by the field parameter.
DOCVARIABLE	Enables you to programmatically insert complex content when this field is updated.
HYPERLINK	Enables you to navigate to another location or to a bookmark.
IF	Compares two values and inserts the text according to the results of the comparison.
INCLUDEPICTURE	Inserts the specified image.
KEYWORDS	Inserts the document keywords or tags.

LASTSAVEDBY	Inserts the name of the last person who modified and saved the document.
MERGEFIELD	Retrieves a value from the bound data source.
NUMPAGES	Inserts the total number of pages.
PAGE	Inserts the number of the page containing the field.
PRINTDATE	Inserts the date and time that a document was last printed.
REVNUM	Inserts the number of document revisions.
SAVEDATE	Inserts a date and time a document was last saved.
SEQ	Provides sequential numbering in the document.
SUBJECT	Inserts the document topic of content.
SYMBOL	Inserts a symbol.
TC	Defines entries for the table of contents.
TIME	Inserts the current time.
TITLE	Inserts the document title.
TOC	Builds a table of contents.

Formatting Options

The following format switches are supported.

- [General Format Switch](#)
- [Date and Time Format Switch](#)
- [Numeric Format Switch](#)

AUTHOR

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [AUTHOR](#)

AUTHOR

Non-MailMerge field

{ AUTHOR }

Inserts information about the document author (person or organization).

Use the DocumentProperties.Creator property to get or set the value of this field in code.

COMMENTS

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [COMMENTS](#)

COMMENTS

Non-MailMerge field

{ COMMENTS }

Inserts the notes about the document content (abstract, table of contents, reference to a graphical representation, etc.).

Use the DocumentProperties.Description property to get or set the value of this field in code.

CREATEDATE

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [CREATEDATE](#)

CREATEDATE

MailMerge Field

{ CREATEDATE [\@ "*format*"] [* MERGEFORMAT]}

Inserts the current date and time in the mail merge document. The *format* parameter is optional. The * MERGEFORMAT switch retains formatting applied to the field.

If you omit the *format* parameter, default formatting is applied. Default formatting is "M/d/yyyy". If you set the FieldOptions.UseCurrentCultureDateTimeFormat option to **true**, the system's Date and Time settings will be used by default.

Field Codes	Field Results
CREATEDATE	4/18/2011
CREATEDATE \@ "dddd, MMMM dd, yyyy HH:mm:ss"	Monday, April 18, 2011 11:25:26

DATE

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [DATE](#)

DATE

Non-MailMerge field

```
{ DATE [ \@ "format" [* MERGEFORMAT]] }
```

Inserts the current date and time. The optional *format* parameter specifies the date format string other than the default system. The [* MERGEFORMAT switch retains formatting applied to the field.

If you omit the *format* parameter, default formatting is applied. The default format is "M/d/yyyy". If you set the FieldOptions.UseCurrentCultureDateTimeFormat option to **true**, the system's Date and Time settings will be used by default.

Field Codes	Field Results
DATE	4/18/2011
DATE \@ "dddd, MMMM dd, yyyy HH:mm:ss"	Monday, April 18, 2011 11:25:26

DOCPROPERTY

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [DOCPROPERTY](#)

DOCPROPERTY

Non-MailMerge field

{ DOCPROPERTY "Name" }

Inserts the value of the document property specified by its name. A name is looked up in the core document property names (Document.DocumentProperties) and subsequently, if not found, in the Document.CustomProperties collection.

Core Document Properties

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(DocumentPropertiesActions.cs)
server.CreateNewDocument();
Document document = server.Document;
document.BeginUpdate();
document.DocumentProperties.Creator = "John Doe";
document.DocumentProperties.Title = "Inserting Custom Properties";
document.DocumentProperties.Category = "TestDoc";
document.DocumentProperties.Description = "This code demonstrates API to modify and display standard document properties";
document.Fields.Create(document.AppendText("\nAUTHOR: ").End, "AUTHOR");
document.Fields.Create(document.AppendText("\nTITLE: ").End, "TITLE");
document.Fields.Create(document.AppendText("\nCOMMENTS: ").End, "COMMENTS");
document.Fields.Create(document.AppendText("\nCREATEDATE: ").End, "CREATEDATE");
document.Fields.Create(document.AppendText("\nCategory: ").End, "DOCPROPERTY Category");
document.Fields.Update();
document.EndUpdate();
```

Visual Basic

```
(DocumentPropertiesActions.vb)
server.CreateNewDocument()
Dim document As Document = server.Document
document.BeginUpdate()
document.DocumentProperties.Creator = "John Doe"
document.DocumentProperties.Title = "Inserting Custom Properties"
document.DocumentProperties.Category = "TestDoc"
document.DocumentProperties.Description = "This code demonstrates API to modify and display standard document properties"
document.Fields.Create(document.AppendText(vbLf & "AUTHOR: ").End, "AUTHOR")
document.Fields.Create(document.AppendText(vbLf & "TITLE: ").End, "TITLE")
document.Fields.Create(document.AppendText(vbLf & "COMMENTS: ").End, "COMMENTS")
document.Fields.Create(document.AppendText(vbLf & "CREATEDATE: ").End, "CREATEDATE")
document.Fields.Create(document.AppendText(vbLf & "Category: ").End, "DOCPROPERTY Category")
document.Fields.Update()
document.EndUpdate()
```

The following names are recognized as core document properties:

Name	Document Property
Author	DocumentProperties.Creator
Category	DocumentProperties.Category
Comments	DocumentProperties.Description

CreateTime	DocumentProperties.Created
Keywords	DocumentProperties.Keywords
LastPrinted	DocumentProperties.LastPrinted
LastSavedBy	DocumentProperties.LastModifiedBy
LastSavedTime	DocumentProperties.Modified
RevisionNumber	DocumentProperties.Revision
Subject	DocumentProperties.Subject
Title	DocumentProperties.Title

Custom Document Properties

A custom document property is specified by its name in double quotes.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#	
<pre>(DocumentPropertiesActions.cs) server.CreateNewDocument(); Document document = server.Document; document.BeginUpdate(); document.Fields.Create(document.AppendText("\nMyNumericProperty: ").End, " document.Fields.Create(document.AppendText("\nMyStringProperty: ").End, "D document.Fields.Create(document.AppendText("\nMyBooleanProperty: ").End, " document.EndUpdate(); document.CustomProperties.Add("MyNumericProperty", 123.45); document.CustomProperties.Add("MyStringProperty", "The Final Answer"); document.CustomProperties.Add("MyBooleanProperty", true); server.CalculateDocumentVariable += DocumentPropertyDisplayHelper.OnCalcul document.Fields.Update();</pre>	
Visual Basic	
<pre>(DocumentPropertiesActions.vb) server.CreateNewDocument() Dim document As Document = server.Document document.BeginUpdate() document.Fields.Create(document.AppendText(vbLf & "MyNumericProperty: ").E document.Fields.Create(document.AppendText(vbLf & "MyStringProperty: ").En document.Fields.Create(document.AppendText(vbLf & "MyBooleanProperty: ").E document.EndUpdate() document.CustomProperties.Add("MyNumericProperty", 123.45) document.CustomProperties.Add("MyStringProperty", "The Final Answer") document.CustomProperties.Add("MyBooleanProperty", True) AddHandler server.CalculateDocumentVariable, AddressOf DocumentPropertyDis document.Fields.Update()</pre>	

DOCVARIABLE

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [DOCVARIABLE](#)

DOCVARIABLE

Mixed field

```
{ DOCVARIABLE "variable name" "argument1" "argument 2"... }
```

Inserts the value of a document variable specified by a text in field argument. Document variables are contained in the DocumentVariableCollection collection object, accessible via the Document.Variables property. Before a field is updated, the Document.CalculateDocumentVariable event is fired allowing you to manually calculate the required value. The CalculateDocumentVariableEventArgs.Arguments provides access to a collection of arguments contained within the field.

Note

You can return a text or the entire IRichEditDocumentServer.Document as a document variable.

A common task is to evaluate DOCVARIABLE fields in a document depending on the variable name and argument and then substitute fields with their values. The resulting document no longer contains fields and its content is fixed. To accomplish this task, you can use two RichEditDocumentServer instances - the first instance loads a document and calls the **MailMerge** method, the second instance handles the IRichEditDocumentServer.CalculateDocumentVariable event. The code

C#

richEditDocumentServer1.MailMerge(richEditDocumentServer2.Document);

Visual Basic

richEditDocumentServer1.MailMerge(richEditDocumentServer2.Document)

produces the final document in the richEditDocumentServer2 instance. Since MailMerge requires a data source for successful execution, you should provide a fake data source. It may be as simple as a collection exposing the [IList](#) interface and containing one item.

HYPERLINK

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [HYPERLINK](#)

HYPERLINK

Non-MailMerge field

```
{ HYPERLINK "location" [switches] }
```

Inserts a hyperlink. When selected, causes control to jump to the location specified by a field argument. That location can be a bookmark or an URL.

Hyperlinks and bookmarks are accessible via the SubDocument.Hyperlinks and SubDocument.Bookmarks properties. When a hyperlink is clicked, the RichEditControl.HyperlinkClick event is fired.

IF

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [IF](#)

IF

Mixed field

```
{ IF Value-1 Operator Value-2 field-argument-1 field-argument-2 }
```


Compares the values designated by Value-1 and Value-2, using the operator designated by Operator. Value-1 and Value-2 can be literals or merge fields. For example, the field code

```
{IF {MERGEFIELD Order} >= 100 "Thanks" "The minimum order is 100 units" }
```

specifies that if the customer order is greater than or equal to 100 units, the result is "Thanks"; but if the customer order is fewer than 100 units, the result is "The minimum order is 100 units".

You can use the **IF** field with **MERGEFIELD** fields. The following statement illustrates this. If the current data record contains different values in HomeAddress and CompanyAddress fields, RichEditControl prints the value of the PresidentName field that is the first set of text in quotation marks. Otherwise, it prints the second set of text in quotation marks that is the value of the OwnerName field.

```
{ IF {MERGEFIELD CompanyAddress} <> {MERGEFIELD HomeAddress} "{MERGEFIELD PresidentName}" "{MERGEFIELD OwnerName}" }
```

 **Important**

Do not use the **IF** field in headers or footers if the condition is based on the **PAGE** and **NUMPAGES** field values. The field value can be incorrect due to limitations of the current layout calculation.

INCLUDEPICTURE

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [INCLUDEPICTURE](#)

INCLUDEPICTURE

Mixed field

```
{ INCLUDEPICTURE "picture's URI" [\d] }
```

Inserts the picture retrieved from the specified URI.

The **\d** switch enables you to not store the image in a document file. For example, the {INCLUDEPICTURE "C:\\TMP\\Mypicture.png" \d} field retains a link to the picture, but the picture itself is not included in the saved document. This switch is useful for storing document templates, which retrieve images from an external source, such as a database.

When a field is updated, the `IUriStreamService.GetStream` method is called. This method calls the `IUriStreamProvider.GetStream` methods of registered providers to obtain a stream of image data. Subsequent calls are made until a successful result is obtained. The order of calls is the order in which providers registered. By default, there is only one provider that enables you to load an image from the specified URI by performing a web request.

This stream is used by internal RichEdit objects to create a `OfficeImage` instance via the `OfficeImage.CreateImage` method.

You can create a class implementing the `IUriStreamProvider` interface and define the `IUriStreamProvider.GetStream` method as required to provide image data for a custom URI, which can be a string in your own arbitrary format.

KEYWORDS

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [KEYWORDS](#)

KEYWORDS

Non-MailMerge field

{ KEYWORDS }

Inserts the document keywords or tags.

Use the DocumentProperties.Keywords property to get or set the value of this field in code.

LASTSAVEDBY

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [LASTSAVEDBY](#)

LASTSAVEDBY

Non-MailMerge field

{ LASTSAVEDBY }

Inserts the name of the last person who modified and saved the document.

Use the DocumentProperties.LastModifiedBy property to get or set the value of this field in code.

MERGEFIELD

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [MERGEFIELD](#)

MERGEFIELD

MailMerge field

```
{ MERGEFIELD "field name" [ switch ] [* MERGEFORMAT] }
```

When the main document is merged with the selected data source, information from the specified data field is inserted in place of the merge field. The [* MERGEFORMAT switch retains the formatting applied to the field.

The \b "**text**" switch specifies the text to be inserted before the MERGEFIELD field if the field is not blank.

The \f "**text**" switch specifies the text to be inserted after the MERGEFIELD field if the field is not blank.

Example:

```
{ MERGEFIELD FirstName \f" " }{ MERGEFIELD MiddleName \f" " }{ MERGEFIELD LastName }
```

If the MiddleName field data is missing (null), then there is only one space between FirstName and LastName:

David Bradley

Otherwise, MiddleName is separated by spaces as required:

David M Bradley

If the merge field is replaced with a DateTime value and no formatting switch is specified, the actual display formatting is dependent on the FieldOptions.UseCurrentCultureDateTimeFormat option. If this option is set to **true**, the system's Date and Time settings will be used. If the FieldOptions.UseCurrentCultureDateTimeFormat is **false** (the default value), DateTime values are displayed using the "M/d/yyyy" formatting.

NUMPAGES

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [NUMPAGES](#)

NUMPAGES

Non-MailMerge field

{ NUMPAGES [switch] }

Inserts the total number of pages.

PAGE

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [PAGE](#)

PAGE

Non-MailMerge field

{ PAGE [switch] }

Inserts the number of the current page.

PRINTDATE

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [PRINTDATE](#)

PRINTDATE

Non-MailMerge field

{ PRINTDATE }

Inserts the date and time that a document was last printed.

Use the DocumentProperties.LastPrinted property to get or set the value of this field in code.

REVNUM

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [REVNUM](#)

REVNUM

Non-MailMerge field

{ REVNUM }

Inserts the number of document revisions.

Use the DocumentProperties.Revision property to get or set the value of this field in code.

SAVEDATE

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [SAVEDATE](#)

SAVEDATE

Non-MailMerge field

{ SAVEDATE }

Inserts the date and time a document was last saved.

Use the DocumentProperties.Modified property to get or set the value of this field in code.

SEQ

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [SEQ](#)

SEQ

Non-MailMerge field

{ SEQ *identifier* [switches] }

Sequentially numbers user-defined lists of items in a document. *Identifier* is the name assigned to the series of items that are to be numbered.

\c Repeats the closest preceding sequence number.

\h Hides the field result.

\n Inserts the next sequence number for the specified item. This is the default.

\r *number* Resets the sequence number to the specified integer number.

SUBJECT

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [SUBJECT](#)

SUBJECT

Non-MailMerge field

{ SUBJECT }

Use the DocumentProperties.Subject property to get or set the value of this field in code.

SYMBOL

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [SYMBOL](#)

SYMBOL

Non-MailMerge field

{ SYMBOL CharNum [switches] }

Inserts a single character or a string of characters.

CharNum - The number that is the ANSI value of the character. You can use either the decimal value or the hexadecimal value in format 0x.

\a - Specifies that the CharNum number is the code for an ANSI character.

\f "fontname" - Specifies the font for the character.

\s number - Specifies the font size in points.

\u - Specifies that the CharNum number is the code for a Unicode character.

TC

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [TC](#)

TC

Non-MailMerge field

{ TC "entry text" [switches] }

Specifies the text and page number for a table of contents entry, which is used by a [TOC](#).

\f identifier The type of items collected in a particular contents list. Use a unique type identifier for each type of list. This switch is required for including into corresponding TOC.

\l "level number" The level of the TC entry. If no level is specified, level 1 is assumed.

TIME

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [TIME](#)

TIME

Non-MailMerge field

```
{ TIME [ \@ "Date-Time Formatting Switch" ] }
```

Inserts the current time.

If you omit the *Date-Time Formatting Switch*, default formatting is applied. Default formatting is "h:mm tt". If you set the `FieldOptions.UseCurrentCultureDateTimeFormat` option to **true**, the system's Date and Time settings will be used by default.

TITLE

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [TITLE](#)

TITLE

Non-MailMerge field

{ TITLE }

Inserts the document title.

Use the DocumentProperties.Title property to get or set the value of this field in code.

TOC

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Field Codes](#) > [TOC](#)

TOC

Non-MailMerge field

{ TOC [switches] }

- Without switches the table of contents is built upon styles with set outline levels. Outline levels assigned to individual paragraphs are disregarded unless the \u switch is specified.
- **\b** Includes entries only from the portion of the document marked by the bookmark named by text in this switch's field-argument.
- **\h** Inserts table of contents entries as hyperlinks.
- **\n "LEVELSTART-LEVELEND"** Omits page numbers from the table of contents. Page numbers are omitted from all levels unless a range of entry levels is specified. For example, { TOC \n 1-1 } omits page numbers from level 1.
- **\u** Uses the applied paragraph outline level.
- **\o "LEVELSTART-LEVELEND"** Builds a table of contents from paragraphs with specified outline levels. For example, { TOC \o "1-3" } lists only paragraphs with outline levels 1 through 3.
- **\t "STYLE,LEVEL, STYLE,LEVEL,..."** Builds a table of contents from paragraphs formatted with specified styles. For example, {TOC \t"Title,1,subtitle,2,customtitle,3"} builds a table of contents from paragraphs formatted with the styles "Title", "subtitle" and "customtitle". The number after each style name indicates the entry level corresponding to that style.
- **\c "SEQ FIELD IDENTIFIER"** Builds a table of contents from items that are numbered by a [SEQ](#). The sequence identifier designated by text in this switch's field-argument shall match the identifier in the corresponding SEQ field.
- **\s "SEQ FIELD IDENTIFIER"** For entries numbered with a SEQ field, adds a prefix to the page number. The prefix depends on the type of entry.
- **\d "SEPARATOR SYMBOL"** When used with \s, defines the separator between sequence and page numbers. The default separator is a hyphen (-).
- **\f "TC IDENTIFIER"** Builds a table of contents from [TC](#). TC field identifier must exactly match the text in this switch's field-argument.
- **\l "LEVELSTART-LEVELEND"** Used with \f key. Includes TC fields that assign entries to one of the specified levels.

Format Switches

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Format Switches](#)

The following format switches are supported.

- [General Format Switch](#)
- [Numeric Format Switch](#)
- [Date and Time Format Switch](#)

See Also

[Field Codes](#)

General Format Switch

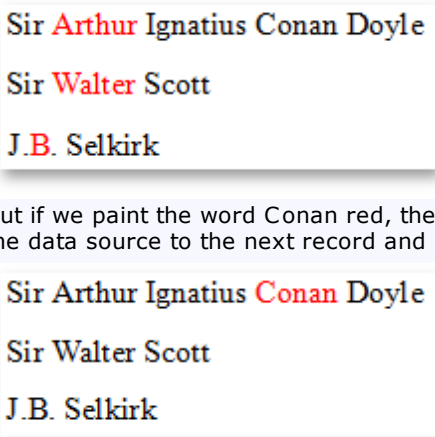
[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Format Switches](#) > [General Format Switch](#)

* ["] switch-argument ["]

A general formatting switch specifies formats for a numeric or text result. If the result type of a field does not correspond to the specified format, this switch has no effect. Quotation marks are required around a switch-argument if it contains a white space; otherwise, they are optional.

The following switch arguments are supported.

Switch Argument	Description
ALPHABETIC	Formats a numeric result as one or more occurrences of an uppercase alphabetic Latin character. Value 1 results in the letter A, value 2 results in the letter B, and so on up to value 26, which results in the letter Z. For values greater than 26, 26 is repeatedly subtracted from the value until the result is 26 or less. The result value determines which letter to use, and the same letter is repeated for each time 26 was subtracted from the original value. For page 27, the field {PAGE * ALPHABETIC} results in "AA", as subtracting 26 from 27 results in the value 1, which is represented by the letter A.
alphabetic	Formats a numeric result as one or more occurrences of a lowercase alphabetic Latin character. Value 1 results in the letter a, value 2 results in the letter b, and so on up to value 26, which results in the letter z. For values greater than 26, 26 is repeatedly subtracted from the value until the result is 26 or less. The result value determines which letter to use, and the same letter is repeated for each time 26 was subtracted from the original value. For page 80, the field {PAGE * alphabetic} results in "bbbb" as subtracting 26 from 80 three times results in the value 2, which is represented by the letter b.
Arabic	Formats a numeric result using Arabic cardinal numerals. For page 123, PAGE * Arabic results in "123".
ArabicDash	Formats a numeric result using Arabic cardinal numerals, with a prefix of "- " and a suffix of " -". For page 123, PAGE * ArabicDash results in "- 123 -".
CardText	Formats a numeric result as lowercase cardinal text. For page 123, PAGE * CardText results in "one hundred twenty-three". Number to text transformation is available for the following languages: English, German, French, Italian, Russian, Swedish, Turkish. The language is determined by the culture of the current thread (System.Threading.Thread.CurrentThread.CurrentCulture).
CIRCLENUM	Formats a numeric result using decimal numbering enclosed in a circle, using the enclosed alphanumeric glyph character for numbers in the range 1 - 20 (Unicode symbols 2460-2473). For non-negative numbers outside this range, formats them as with ARABIC. Note that the font used to display a field should have corresponding glyphs in the specified positions.
DBCHAR	Formats a numeric result using double-byte Arabic numbering.
DBNUM1	Formats a numeric result using ideographic numbers for each digit in decimal representation. The Zero (0) is displayed using the character U+3007 (IDEOGRAPHIC NUMBER ZERO), while other digits are displayed using Chinese numbers from 1 to 9.
DollarText	Formats a numeric result in the following form: integer-part-as-cardinal-text and nn/100. The fractional part is rounded to two decimal places and is formatted using Arabic cardinal numerals. The number 123.456 will be displayed as "one hundred twenty-three and 46/100"
GB1	Formats a numeric result using numbers with full stop Unicode characters. Numbers from 1 to 20 correspond to symbols from \u2488 to \u249B.
GB2	Formats a numeric result using parenthesized numbers. Numbers from 1 to 20 correspond to Unicode symbols from \u2474 to \u2487.

GB3	Formats a numeric result using parenthesized ideographs. Numbers from 1 to 10 correspond to Unicode symbols from \u3220 to \u3229.
Hex	Formats the numeric result using uppercase hexadecimal digits. For page 44, PAGE * Hex results in "2C"
KANJINUM1	Formats a numeric result using Japanese sequential digital ideographs.
MERGEFORMAT	<p>When the field is updated and you apply formatting to the field result, that formatting is retained the next time the field is updated. Usually this works, but since formatting is dependent on the parts of a field result you apply formatting to, there are situations when this switch is unreliable.</p> <p>The following picture illustrates how the MERGEFORMAT switch works. The MERGEFIELD is bound to the list of writer names. After the field is updated, it shows the first name in the list. We paint the word Arthur red, and then move to the next data record. Red highlighting is retained.</p>  <p>But if we paint the word Conan red, the highlighting will be lost after we navigate the data source to the next record and update the field.</p>
Ordinal	<p>Formats a numeric result using lowercase ordinal Arabic numerals. Ordinal conversion is available for the following languages: English. German, French, Italian, Russian, Swedish, Turkish. The language is determined by the culture of the current thread</p> <p>(System.Threading.Thread.CurrentThread.CurrentCulture)</p>
OrdText	<p>Formats a numeric result as lowercase ordinal text. The fractional part is used to round off the whole number part and then discarded. Ordinal conversion is available for the following languages: English. German, French, Italian, Russian, Swedish, Turkish. The language is determined by the culture of the current thread (System.Threading.Thread.CurrentThread.CurrentCulture).</p>
Roman	Formats a numeric result using uppercase Roman numerals.
roman	Formats a numeric result using lowercase Roman numerals.
SBCHAR	Formats a numeric result using common, single-byte Arabic numbering.
ZODIAC1	Formats a numeric result using sequential numerical ideographs.
ZODIAC2	Formats a numeric result using sequential numerical ideographs.
ZODIAC3	Formats a numeric result using sequential numerical ideographs.

Numeric Format Switch

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Format Switches](#) > [Numeric Format Switch](#)

\# ["] switch-argument ["]

A numeric switch-argument is like a numeric format string. If a switch-argument contains spaces, enclose it in quotation marks. You can combine the following items to construct a format string.

Item	Usage
#	Specifies the numeric places to display in the result. If the result does not include a digit in that place, a space is displayed.
0 (zero)	Specifies the numeric places to display in the result. If the result does not include a digit in that place, a 0 (zero) is displayed.
. (decimal point)	Determines the decimal point position. The actual decimal symbol is dependent on the local Culture. You can specify it in the Regional settings (Control Panel).
, (digit grouping symbol)	Separates a series of three digits. The actual symbol is dependent on the local Culture. You can specify it in the Regional settings (Control Panel).
x	Drops digits to the left of the "x" placeholder. If the placeholder is to the right of the decimal point, the result is rounded to that position.
- (minus sign)	Adds a minus sign to a negative result, or adds a space if the result is positive or 0 (zero).
+ (plus sign)	Adds a plus sign to a positive result, a minus sign to a negative result, or a space if the result is 0 (zero).
other symbol(s)	Includes the specified character in the result.
'text'	Adds text to the result. The text can contain any symbols, special symbols mentioned above are not interpreted.
"positive; negative; zero"	You can use combinations for displaying positive, negative and zero values differently - just use a semicolon between switch arguments. For example, { MERGEFIELD Amount \# \$#,##0.00;- \$#,##0.00;\$0 } displays positive values using \$#,##0.00, negative values using - \$#,##0.00, and zero values using \$0.

Date and Time Format Switch

[Office File API](#) > [Word Processing Document API](#) > [Fields](#) > [Format Switches](#) > [Date and Time Format Switch](#)

\@ ["] **switch-argument** ["]

A date-and-time format switch specifies the format of a date or time result. Use "M" for the month, "d" for the Day. For the year, use "yy" or "yyyy". For the Hours, use "h" or "hh" for the hour, "m" or "mm" for the minute, and "am/pm" to include the AM/PM designation. To include other text inside the result, enclose it in single quote marks.

Specifier	Description
d	Displays the day as a number without a leading zero for single-digit days.
dd	Displays the day as a number with a leading zero for single-digit days.
ddd	Displays the day of the week in its abbreviated form according to the current culture.
dddd	Displays the full name of the day of the week according to the current culture.
M	Displays the month as a number without a leading zero for single-digit months.
MM	Displays the month as a number with a leading zero for single-digit months.
MMM	Displays the month in its abbreviated form according to the current culture.
MMMM	Displays the full name of the month according to the current culture.
yy	Displays the year number as two digits with a leading zero for years 0 - 9.
yyyy	Displays the year number as four digits
h or H	Displays the hour as a number without a leading zero for single-digit hours.
hh or HH	Displays the hour as a number with a leading zero for single-digit hours.
m	Displays the minutes as a number without a leading zero for single-digit minutes.
mm	Displays the minutes as a number with a leading zero for single-digit minutes.
am/pm or AM/PM	Displays time using am/pm or AM/PM notation.
ss	Displays the seconds as a number with a leading zero for single-digit seconds.

Examples:

1. Field code {DATE \@ "'Today is 'MMM. d, yyyy'"} results in **Today is Dec. 29, 2011**
2. Field code {DATE \@ "'Current time is 'HH:mm:ss'"} results in **Current time is 17:13:15**

Mail Merge

[Office File API](#) > [Word Processing Document API](#) > [Mail Merge](#)

Topics in this section:

- [Mail Merge](#)
- [Master-Detail Report](#)

Mail Merge

[Office File API](#) > [Word Processing Document API](#) > [Mail Merge](#) > [Mail Merge](#)

Perform the following steps to create the mail merge document:

1. [Provide a Data Source](#)
2. [Load a Document Template](#)
3. [Merge and Save the Document](#)

Provide a Data Source

Use the RichEditMailMergeOptions.DataSource property to set the data source to use in a mail merge. If the data source contains multiple data tables, use the MailMergeOptions.DataMember property to define a specific data member. RichEditDocumentServer supports the following data source types:

- **System.Collections.IList;**
- **System.Collections.ArrayList;**
- **System.Data.DataTable.**

C#

```
server.Options.MailMerge.DataSource = new SampleData();
server.Options.MailMerge.ViewMergedData = true;
```


Visual Basic

```
server.Options.MailMerge.DataSource = New SampleData()
server.Options.MailMerge.ViewMergedData = True
```

Load a Document Template

The template is a document containing fields, which refer to a specified data source's data column names. Call the FieldCollection.Create method to insert the desired field to the given document position in code. Depending on the content to be inserted, use one of the following fields:

- [MERGEFIELD](#) - to insert plain text;
- [DOCVARIABLE](#) - to insert formatted content;
- [INCLUDEPICTURE](#) - to insert images. Use the INCLUDEPICTURE field as follows: {INCLUDEPICTURE "{MERGEFIELD PictureLocation}" \d}, where **PictureLocation** is the path to the desired picture. Make sure that images have the same name as the field's contents in the database (for example, as the "FirstName" field) if the template requires mail merging different images, and insert the INCLUDEPICTURE as follows: { INCLUDEPICTURE "PicturesLocation\ \{ MERGEFIELD FieldName }.jpg *MERGEFORMAT \d }.

 Tip

Use the IUriStreamProvider to insert pictures from a data base. Refer to the [E4164](#) code example for details.

Merge and Save the Document

Use the following API to set additional merging options (the number of records, merged ranges delimitation, etc.) and merge the document:

API	Description
RichEditControlOptionsBase.MailMerge	Provides access to the default mail merge options.
Document.CreateMailMergeOptions	Initializes an MailMergeOptions instance, which contains additional mail merge options.
MailMergeOptions.FirstRecordIndex	Gets or sets the record index from which the merge starts.
MailMergeOptions.LastRecordIndex	Gets or sets the record index at which the merge finishes.
MailMergeOptions.MergeMode	Gets or sets how the merged ranges are delimited in the resulting document.

Document.MailMerge	Merges the current document using the specified options, and sends the result to the specified document.
--------------------	--

After the mail merge is finished, you can send the document to further processing to another RichEditDocumentServer instance, or save it to the file.

C#

```
using DevExpress.XtraRichEdit.API.Native;
...
MailMergeOptions myMergeOptions = server.Document.CreateMailMergeOptions();
myMergeOptions.MergeMode = MergeMode.NewSection;
server.Document.MailMerge(myMergeOptions, "Result.docx", DocumentFormat.OpenXml);
```

Visual Basic

```
Imports DevExpress.XtraRichEdit.API.Native
...
Dim myMergeOptions As MailMergeOptions = server.Document.CreateMailMergeOptions()
myMergeOptions.MergeMode = MergeMode.NewSection
server.Document.MailMerge(myMergeOptions, "Result.docx", DocumentFormat.OpenXml)
```


Mail Merge Events

Additionally, the RichEditDocumentServer provides the following events allowing you to control specific mail merging steps:

Event	Description
RichEditDocumentServer.MailMergeStarted	Fires before mail merge starts.
RichEditDocumentServer.MailMergeFinished	Fires when mail merge is completed.
RichEditDocumentServer.MailMergeRecordStarted	Fires before each data record is merged with the document in the mail merge process.
RichEditDocumentServer.MailMergeRecordFinished	Fires after each data record is merged with the document in the mail merge process.

Master-Detail Mail Merge Documents

The master-detail report requires creating a RichEditDocumentServer instance as an intermediate document on each data level that is inserted in place of the DOCVARIABLE field. Refer to the [Master-Detail Report](#) topic for details on how to create a master-detail mail merge document.

 Tip

You can transform a mail merge template to a self-contained document in the same RichEditDocumentServer instance by using current data. Refer to the [How to replace document fields with their values](#) example for details.

See Also
[Master-Detail Report](#)

Master-Detail Report

[Office File API](#) > [Word Processing Document API](#) > [Mail Merge](#) > [Master-Detail Report](#)

This article describes how the [Mail Merge](#) feature enhanced with [DOCVARIABLE](#) specifics allows users to accomplish complex tasks such as creating Master-Detail reports.

In this example, the report is generated using four templates: **Main**, **Master**, **Detail**, and **DetailDetail** containing DOCVARIABLE fields. A RichEditDocumentServer instance is created on each data level for holding the corresponding template and for calculating in the RichEditDocumentServer.CalculateDocumentVariable event handler. As a result, all intermediate documents are combined into a single report, which is passed to the parent document to substitute the initial DOCVARIABLE field.

The following image displays the result:

This document is created by using the Master-Detail MailMerge feature of the Rich Edit Document Server. Data are obtained from the ITypedList datasource. The report has to be inserted in place of the DOCVARIABLE field below:

EXOTIC LIQUIDS

Chai

ORDER ID	QUANTITY
<u>rbpfvqnffs</u>	33
<u>oy4lsog3lmm</u>	28
<u>jxycfaqhw0j</u>	26

Chang

ORDER ID	QUANTITY
<u>ppoiiaqkpgt</u>	62
<u>4ccgem5virl</u>	46
<u>nbcipamcxvf</u>	92

Aniseed Syrup

ORDER ID	QUANTITY
<u>j0adh5tdfha</u>	14
<u>isqabzymg0b</u>	95
<u>3iseoiygemr</u>	59

The steps below provide more detailed information about how to process each part of a master-detail report.

- [Main Document](#)
- [Master Template](#)
- [Detail Template](#)
- [Detail-Detail Template](#)

Main Template

The main template is the document without MERGEFIELD fields. It may contain a logo, standard header and footer, and all the parts required for a corporate design and layout. The template's DOCVARIABLE field indicates where the **Master** template is inserted. Use the FieldCollection.Create method to insert a field into the desired position in the document.

Initiate the mail merge process with empty data. Assign the control to a mock data source and call the Document.MailMerge method with the resulting control's document or a file set as the destination (in this example, the result is passed to the **resultRichEdit**).

[Show Me](#)

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4718>.

C#

```
(MergeProcessor.cs)
// Start the process by merging main template into the document contained within the resultRichEdit server
public void Start()
{
    // Since the main template contains no merge fields requiring no merge data, provide a fake data source
    // Otherwise mailmerge will not start.
    mainRichEdit.Options.MailMerge.DataSource = fakeDataSource;
    // Trigger the multistage process. After the first mailmerge the CalculateDocumentVariable event
    //for the resultRichEdit server fires.
    mainRichEdit.MailMerge(resultRichEdit.Document);
    resultRichEdit.SaveDocument("result.docx", DocumentFormat.OpenXml);
}
```

Visual Basic

```
(MergeProcessor.vb)
' Start the process by merging main template into the document contained within the resultRichEdit server
Public Sub Start()
    ' Since the main template contains no merge fields requiring no merge data, provide a fake data source
    ' Otherwise mailmerge will not start.
    mainRichEdit.Options.MailMerge.DataSource = fakeDataSource
    ' Trigger the multistage process. After the first mailmerge the CalculateDocumentVariable event
    'for the resultRichEdit server fires.
    mainRichEdit.MailMerge(resultRichEdit.Document)
    resultRichEdit.SaveDocument("result.docx", DocumentFormat.OpenXml)
End Sub
```

Master Template

The destination control receives the **Main** document and fires the RichEditDocumentServer.CalculateDocumentVariable event. Handle this event the following way to create a master part and insert it into the document:

1. Make sure that the event is fired for the required field (by checking the CalculateDocumentVariableEventArgs.VariableName value).
2. Create a new RichEditDocumentServer instance to create a report and pass it to the event sender using the CalculateDocumentVariableEventArgs.Value property.
3. Subscribe the created document server instance to the RichEditControl.CalculateDocumentVariable event to process the detail part of the report.
4. Call the Document.MailMerge method to merge the **Master** template and obtain the source document with the corresponding data.
5. Pass the document to insert to the **e.Value** property.
6. Set the **Handled** property to **true**. Otherwise, the previous step is ignored.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4718>

C#

```
(MergeProcessor.cs)
// Second stage. For each Supplier ID create a detailed document that will
void resultRichEdit_CalculateDocumentVariable(object sender, CalculateDocu
{
    if (e.VariableName == "Supplier") {
        // Create a text engine to process a document after the mail merge
        RichEditDocumentServer richServerMaster = new RichEditDocumentServ
        // Provide a procedure for further processing
        richServerMaster.CalculateDocumentVariable += richServerMaster_Cal
        // Create a merged document using the Supplier template. The docum
        // The CalculateDocumentVariable event for the richServerMaster fi
        supplierRichEdit.MailMerge(richServerMaster);
        richServerMaster.CalculateDocumentVariable -= richServerMaster_Cal
        // Return the document to insert.
        e.Value = richServerMaster;
        // Required to use e.Value. Otherwise it will be ignored.
        e.Handled = true;
    }
}
```

Visual Basic

```
(MergeProcessor.vb)
' Second stage. For each Supplier ID create a detailed document that will
Private Sub resultRichEdit_CalculateDocumentVariable(ByVal sender As Objec
    If e.VariableName = "Supplier" Then
        ' Create a text engine to process a document after the mail merge.
        Dim richServerMaster As New RichEditDocumentServer()
        ' Provide a procedure for further processing
        AddHandler richServerMaster.CalculateDocumentVariable, AddressOf r
        ' Create a merged document using the Supplier template. The docume
        ' The CalculateDocumentVariable event for the richServerMaster fir
        supplierRichEdit.MailMerge(richServerMaster)
        RemoveHandler richServerMaster.CalculateDocumentVariable, AddressC
        ' Return the document to insert.
        e.Value = richServerMaster
        ' Required to use e.Value. Otherwise it will be ignored.
        e.Handled = True
    End If
End Sub
```

Detail Template

After the mail merge, the document created from the **Master** template is loaded into the RichEditDocumentServer instance, and the RichEditDocumentServer.CalculateDocumentVariable event fires. Handle the event to compose the **Detail** template:

1. Make sure that the event is fired for the required field (by checking the CalculateDocumentVariableEventArgs.VariableName value);
2. Create new RichEditDocumentServer instance.
3. This template contains a master-detail table header. The resulting table is composed of separate tables. Use the MergeMode.JoinTables option to make the

tables continuous. When not using tables, assigning this value to the `MailMergeOptions.MergeMode` property results in an incorrect layout.

4. Subscribe to the `Document.CalculateDocumentVariable` event of the current `RichEditDocumentServer` instance for further processing. This step is required if the **Detail** template contains any `DOCVARIABLE` fields. In this example, the `Document.CalculateDocumentVariable` event is used to insert and format the **UnitPrice** field.
5. Call the `Document.MailMerge` method to merge the **Detail** template.
6. Pass the document to insert to the **e.Value** property.
7. Set the **Handled** property to **true**. Otherwise, the previous step is ignored.

🔗 Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4718>

C#	
<pre>(MergeProcessor.cs) // Third stage. For each Product ID create a detailed document that will be void richServerMaster_CalculateDocumentVariable(object sender, CalculateDoc { int currentSupplierID = GetID(e.Arguments[0].Value); if (currentSupplierID == -1) return; if (supplierID != currentSupplierID) { // Get data source that contains products for the specified supplier dataDetailedForProducts = GetProductsDataFilteredbySupplier(current supplierID = currentSupplierID; } if (e.VariableName == "Product") { // Create a text engine to process a document after the mail merge RichEditDocumentServer richServerDetail = new RichEditDocumentServ // Specify data source for mail merge. MailMergeOptions options = productRichEdit.CreateMailMergeOptions(options.DataSource = dataDetailedForProducts; // Specify that the resulting table should be joined with the head // Do not specify this option if calculated fields are not within options.MergeMode = MergeMode.JoinTables; // Provide a procedure for further processing. richServerDetail.CalculateDocumentVariable += richServerDetail_Cal // Create a merged document using the Product template. The docume // The CalculateDocumentVariable event for the richServerDetail fi productRichEdit.MailMerge(options, richServerDetail); richServerDetail.CalculateDocumentVariable -= richServerDetail_Cal // Return the document to insert. e.Value = richServerDetail; // This setting is required for inserting e.Value into the source e.Handled = true; } }</pre>	

Visual Basic

```

(MergeProcessor.vb)
' Third stage. For each Product ID create a detailed document that will be
Private Sub richServerMaster_CalculateDocumentVariable(ByVal sender As Obj
    Dim currentSupplierID As Integer = GetID(e.Arguments(0).Value)
    If currentSupplierID = -1 Then
        Return
    End If
    If supplierID <> currentSupplierID Then
        ' Get data source that contains products for the specified supplie
        dataDetailedForProducts = GetProductsDataFilteredbySupplier(curren
        supplierID = currentSupplierID
    End If
    If e.VariableName = "Product" Then
        ' Create a text engine to process a document after the mail merge.
        Dim richServerDetail As New RichEditDocumentServer()
        ' Specify data source for mail merge.
        Dim options As MailMergeOptions = productRichEdit.CreateMailMergeO
        options.DataSource = dataDetailedForProducts
        ' Specify that the resulting table should be joined with the heade
        ' Do not specify this option if calculated fields are not within t
        options.MergeMode = MergeMode.JoinTables
        ' Provide a procedure for further processing.
        AddHandler richServerDetail.CalculateDocumentVariable, AddressOf r
        ' Create a merged document using the Product template. The documen
        ' The CalculateDocumentVariable event for the richServerDetail fir
        productRichEdit.MailMerge(options, richServerDetail)
        RemoveHandler richServerDetail.CalculateDocumentVariable, AddressC
        ' Return the document to insert.
        e.Value = richServerDetail
        ' This setting is required for inserting e.Value into the source d
        e.Handled = True
    End If
End Sub

```

Detail-Detail Template

This is the final merging part. Handle the RichEditDocumentServer.CalculateDocumentVariable event to compose this part as follows:

1. Make sure that the event is fired for the required field (by checking the CalculateDocumentVariableEventArgs.VariableName value).
2. Create new RichEditDocumentServer and MailMergeOptions instances.
3. This template also contains a master-detail table header. Set the MailMergeOptions.MergeMode property to MergeMode.JoinTables to make the tables continuous. When not using tables, assigning this value to the **MergeMode** property results in an incorrect layout.
4. Provide the data source for merging. In this example, the data is filtered by supplier, then retrieved and passed as the RichEditMailMergeOptions.DataSource property value.

5. Call the Document.MailMerge method with given mail merge options to merge the **Detail-Detail** template.
6. Pass the document to insert to the **e.Value** property.
7. Set the **Handled** property to **true**. Otherwise, the previous step is ignored.

☛ Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4718>

C#	
<pre>(MergeProcessor.cs) // Fourth stage. For each Order ID create a detailed document that will be // This is the final stage and the Product.Orders template does not contain void richServerDetail_CalculateDocumentVariable(object sender, CalculateDoc { int currentProductID = GetID(e.Arguments[0].Value); if (currentProductID == -1) return; if (productID != currentProductID) { // Get data source that contains orders for the specified product. // The data source is obtained from the data already filtered by s dataDetailedForOrders = GetOrderDataFilteredbyProductAndSupplier(c productID = currentProductID; } if (e.VariableName == "OrderDetails") { RichEditDocumentServer richServerDetailDetail = new RichEditDocume MailMergeOptions options = ordersRichEdit.CreateMailMergeOptions() options.DataSource = dataDetailedForOrders; options.MergeMode = MergeMode.JoinTables; ordersRichEdit.MailMerge(options, richServerDetailDetail); e.Value = richServerDetailDetail; e.Handled = true; } }</pre>	
Visual Basic	

```
(MergeProcessor.vb)
' Fourth stage. For each Order ID create a detailed document that will be
' This is the final stage and the Product.Orders template does not contain
Private Sub richServerDetail_CalculateDocumentVariable(ByVal sender As Obj
    Dim currentProductID As Integer = GetID(e.Arguments(0).Value)
    If currentProductID = -1 Then
        Return
    End If
    If productID <> currentProductID Then
        ' Get data source that contains orders for the specified product.
        ' The data source is obtained from the data already filtered by su
        dataDetailedForOrders = GetOrderDataFilteredbyProductAndSupplier(c
        productID = currentProductID
    End If
    If e.VariableName = "OrderDetails" Then
        Dim richServerDetailDetail As New RichEditDocumentServer()
        Dim options As MailMergeOptions = ordersRichEdit.CreateMailMergeOp
        options.DataSource = dataDetailedForOrders
        options.MergeMode = MergeMode.JoinTables
        ordersRichEdit.MailMerge(options, richServerDetailDetail)
        e.Value = richServerDetailDetail
        e.Handled = True
    End If
End Sub
```

Printing

[Office File API](#) > [Word Processing Document API](#) > [Printing](#)

This document outlines the techniques used to print from the RichEditDocumentServer. It consists of the following sections:

- [Print with the Default Printer](#)
- [Change Printer Settings](#)
- [DevExpress Printing Library \(WinForms\)](#)
- [RichEditDocumentXpfPrinter \(WPF\)](#)
- [Export to PDF \(ASP.NET\)](#)

Print with the Default Printer

RichEditDocumentServer allows you to print a document with the default settings without end-user interaction. Use the RichEditDocumentServer.Print method as shown below to complete the task:

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(BasicActions.cs)
server.Document.AppendDocumentContent("Documents\\Grimm.docx", DocumentFormat.OpenXml);
server.Print();

Visual Basic

(BasicActions.vb)
server.Document.AppendDocumentContent("Documents\Grimm.docx", DocumentFormat.OpenXml)
server.Print()

Change Printer Settings

Call the RichEditDocumentServer.Print method with the passed *System.Drawing.Printing.PrinterSettings* instance to specify the desired printer options (the range of pages to print, the number of copies, etc.) to print a document.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T590908>.

C#

(XtraForm1.cs)
PrinterSettings printerSettings = new PrinterSettings();
//Set the document pages to print:
printerSettings.FromPage = 2;
printerSettings.ToPage = 3;
//Specify the number of copies:
printerSettings.Copies = 2;
//Print the document:
server.Print(printerSettings);

Visual Basic

```
(XtraForm1.vb)
Dim printerSettings As New PrinterSettings()
'Set the document pages to print:
printerSettings.FromPage = 2
printerSettings.ToPage = 3
'Specify the number of copies:
printerSettings.Copies = 2
'Print the document:
server.Print(printerSettings)
```

DevExpress Printing Library (WinForms)

To set options that are unavailable from RichEditDocumentServer directly, use the DevExpress Printing Library. This library has the PrintableComponentLink class, which allows you to change required printing settings (e.g., use the specific printer or disable showing error messages). For this, use the API from the table below as shown in the following code snippet.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4846>

C#	
<pre>(Form1.cs) private static void PrintViaLink(RichEditDocumentServer srv) { if (!srv.IsPrintingAvailable) return; PrintableComponentLink link = new PrintableComponentLink(new PrintingsS link.Component = srv; // Disable warnings. link.PrintingSystem.ShowMarginsWarning = false; link.PrintingSystem.ShowPrintStatusDialog = false; // Find a printer containing 'Canon' in its name. string printerName = String.Empty; for (int i = 0; i < PrinterSettings.InstalledPrinters.Count; i++) { string pName = PrinterSettings.InstalledPrinters[i]; if (pName.Contains("Canon")) { printerName = pName; break; } } // Print to the specified printer. link.Print(printerName); }</pre>	
Visual Basic	

```
(Form1.vb)
Private Shared Sub PrintViaLink(ByVal srv As RichEditDocumentServer)
    If Not srv.IsPrintingAvailable Then
        Return
    End If
    Dim link As New PrintableComponentLink(New PrintingSystem())
    link.Component = srv
    ' Disable warnings.
    link.PrintingSystem.ShowMarginsWarning = False
    link.PrintingSystem.ShowPrintStatusDialog = False
    ' Find a printer containing 'Canon' in its name.
    Dim printerName As String = String.Empty
    For i As Integer = 0 To PrinterSettings.InstalledPrinters.Count - 1
        Dim pName As String = PrinterSettings.InstalledPrinters(i)
        If pName.Contains("Canon") Then
            printerName = pName
            Exit For
        End If
    Next i
    ' Print to the specified printer.
    link.Print(printerName)
End Sub
```

RichEditDocumentXpfPrinter (WPF)

To print a document loaded in the RichEditDocumentServer, create a System.Windows.Documents.FixedDocument using the RichEditDocumentXpfPrinter.CreateFixedDocument method and utilize the **System.Windows.Controls.PrintDialog** class to print the fixed document.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E3862>

.


C#	
----	--

```
(MainWindow.xaml.cs)
RichEditDocumentServer srv = new RichEditDocumentServer();
srv.LoadDocument("test.docx");
FixedDocument document = RichEditDocumentXpfPrinter.CreateFixedDocument(srv);
PrintDialog pDialog = new PrintDialog();
PrintQueueCollection queues = new PrintServer().GetPrintQueues(new[] { EnumeratedPrintQueueTypes.Connections});
System.Collections.IEnumerator localPrinterEnumerator = queues.GetEnumerator();
PrintQueue printQueue = null;
do
{
    if (!localPrinterEnumerator.MoveNext())
        break;
    printQueue = (PrintQueue)localPrinterEnumerator.Current;
}
while (!printQueue.FullName.Contains("Canon"));
if (printQueue != null)
{
    pDialog.PrintQueue = printQueue;
    pDialog.PrintDocument(document.DocumentPaginator, string.Empty);
}
```

Visual Basic
<pre>(MainWindow.xaml.vb) Dim srv As New RichEditDocumentServer() srv.LoadDocument("test.docx") Dim document As FixedDocument = RichEditDocumentXpfPrinter.CreateFixedDocument(srv) Dim pDialog As New PrintDialog() Dim queues As PrintQueueCollection = (New PrintServer()).GetPrintQueues(EnumeratedPrintQueueTypes.Connections) Dim localPrinterEnumerator As System.Collections.IEnumerator = queues.GetEnumerator() Dim printQueue As PrintQueue = Nothing Do If Not localPrinterEnumerator.MoveNext() Then Exit Do End If printQueue = DirectCast(localPrinterEnumerator.Current, PrintQueue) Loop While Not printQueue.FullName.Contains("Canon") If printQueue IsNot Nothing Then pDialog.PrintQueue = printQueue pDialog.PrintDocument(document.DocumentPaginator, String.Empty) End If</pre>

Export to PDF (ASP.NET)

Use the RichEditDocumentServer.ExportToPdf method which sends PDF output to a stream, as illustrated in the following code snippet:

 Tip
Create the PdfExportOptions instance, set its PdfExportOptions.ShowPrintDialogOnOpen to true and pass this object to the RichEditDocumentServer.ExportToPdf method to open the Print dialog when opening the file.
C#

```
RichEditDocumentServer documentServer = new RichEditDocumentServer();  
// Your code here to create, load or modify a document in the RichEditDocu  
using (MemoryStream stream = new MemoryStream()) {  
    PdfExportOptions options = new PdfExportOptions();  
    options.ShowPrintDialogOnOpen = true;  
    documentServer.ExportToPdf(stream, options);  
    stream.Seek(0, SeekOrigin.Begin);  
    Response.ContentType = "application/pdf";  
    Response.AddHeader("content-disposition", "inline; filename=ExportedPd  
    stream.CopyTo(Response.OutputStream);  
    Response.End();  
}
```

Visual Basic

```
Dim documentServer As New RichEditDocumentServer()  
' Your code here to create, load or modify a document in the RichEditDocum  
Using stream As New MemoryStream()  
    Dim options As New PdfExportOptions()  
    options.ShowPrintDialogOnOpen = True  
    documentServer.ExportToPdf(stream, options)  
    stream.Seek(0, SeekOrigin.Begin)  
    Response.ContentType = "application/pdf"  
    Response.AddHeader("content-disposition", "inline; filename=ExportedPd  
    stream.CopyTo(Response.OutputStream)  
    Response.End()  
End Using
```

Important

If an application is hosted on **Microsoft Azure**, set the `AzureCompatibility.Enable` property to **true**.

See Also

[Printing in Examples Section](#)

Export to PDF

[Office File API](#) > [Word Processing Document API](#) > [Export to PDF](#)

Call the RichEditDocumentServer.ExportToPdf method to export the document to the PDF format. Use the PdfExportOptions object's properties to define the required export options, as shown in the following code:

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ExportActions.cs)
server.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml);
//Specify export options:
PdfExportOptions options = new PdfExportOptions();
options.DocumentOptions.Author = "Mark Jones";
options.Compressed = false;
options.ImageQuality = PdfJpegImageQuality.Highest;
//Export the document to the stream:
using (FileStream pdfFileStream = new FileStream("Document_PDF.pdf", FileMode.Create))
{
    server.ExportToPdf(pdfFileStream, options);
}
System.Diagnostics.Process.Start("Document PDF.pdf");
```

Visual Basic

```
(ExportActions.vb)
server.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml)
'Specify export options:
Dim options As New PdfExportOptions()
options.DocumentOptions.Author = "Mark Jones"
options.Compressed = False
options.ImageQuality = PdfJpegImageQuality.Highest
'Export the document to the stream:
Using pdfFileStream As New FileStream("Document_PDF.pdf", FileMode.Create)
    server.ExportToPdf(pdfFileStream, options)
End Using
System.Diagnostics.Process.Start("Document PDF.pdf")
```

Sections below provide more detailed information about the PDF export specifics.

PDF Document Versions

The PDF export engine produces PDF documents of the following versions depending on specified export options:

- A **PDF 1.4** document is generated by default and is supported by the Adobe Acrobat version **5.0** or later.
- A **PDF 1.6** document is generated if you enable [encryption](#) in PDF options and set the PdfPasswordSecurityOptions.EncryptionLevel property to PdfEncryptionAlgorithm.AES128 or PdfEncryptionAlgorithm.ARC4. This document version is supported by Adobe Acrobat **7.0** or later.
- A **PDF 1.7** document is generated if you enable [encryption](#) in PDF options and set the PdfPasswordSecurityOptions.EncryptionLevel property to

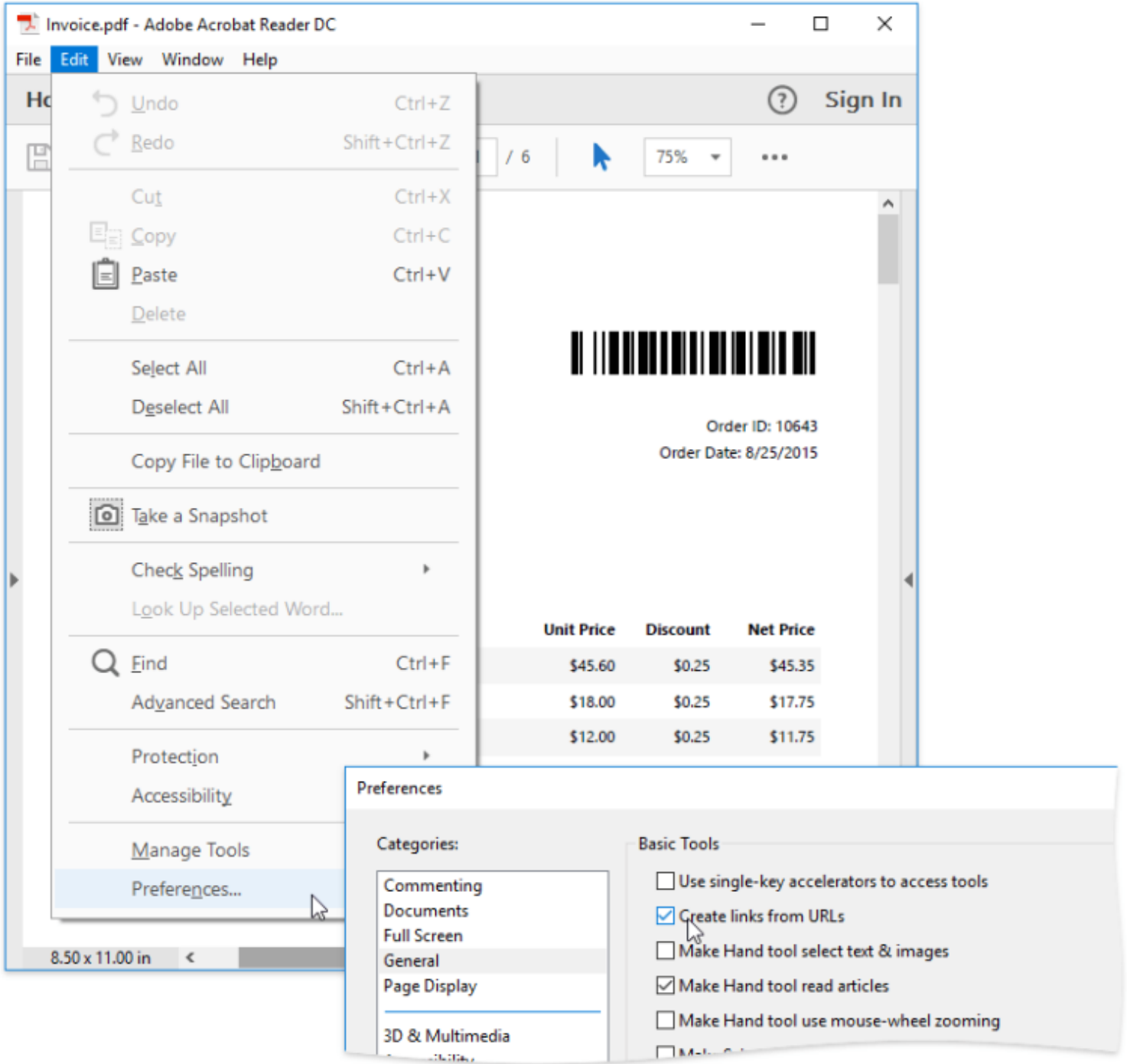
PdfEncryptionAlgorithm.AES256. This document version is supported by Adobe Acrobat **9.0** or later.

PDF Cross-Compatibility

PDF generation does not require installing third-party components on an end-user's machine.

- When including Far-Eastern and Arabic fonts into PDF, make sure that the included fonts contain all necessary characters.
- Document hyperlinks will be active in PDF only if the **Create links from URLs** setting is enabled in Adobe® Reader installed on the target machine.

To access this option in the program's **Edit** menu, click **Preferences** and in the invoked dialog, switch to **General**.



The document pages' background is filled with the color specified by the XtraReport.PageColor property. If you want to add a background image to the PDF file, set this property to **Transparent**.

The PDF options are provided by the PdfExportOptions class. These options can be accessed via a report's ExportOptions.Pdf property and are detailed in the following sections.

PDF/A Options

The following PDF/A specifications are supported.

- **PDF/A-1b (ISO 19005-1)**

- **PDF/A-2b (ISO 19005-2:2011)**
- **PDF/A-3b (ISO 19005-3:2012)**

To make a document compatible with the PDF/A specification, use the following options.

- PdfExportOptions.PdfACompatibility
- PdfExportOptions.AdditionalMetadata

If the PdfExportOptions.PdfACompatibility property is set to PdfACompatibility.None (the default value), the resulting document will conform to the **ISO 32000-1:2005** standard without any restrictions.

For a code sample, refer to the following example online: [How to export a report to ZUGFeRD](#).

For the current versions of the library, consider the following restrictions associated with PDF/A compatibility.

- All PDF/A versions implicitly prohibit encryption.
- All fonts that are used in PDF/A documents should be embedded.
- The **PDF/A-1b** and **PDF/A-2b** standards do not support attachments.
- The **PDF/A-1b** standard does not support transparency, and the alpha channel in images will be ignored.

To check the validity of PDF export options, use the PdfExportOptions.Validate method that returns a list of any detected inconsistencies.

Document Options

- PdfExportOptions.DocumentOptions
- PageByPageExportOptionsBase.PageRange
- PdfExportOptions.ShowPrintDialogOnOpen

Security Options

- Use the PdfExportOptions.PasswordSecurityOptions property to protect the resulting PDF file from being read, copied, printed and/or edited by specifying passwords for these actions.

Important

PDF passwords are saved with report definitions in clear text. Make sure that only trusted parties have access to report definition files.

It is also possible to enable text access to a document for screen reader devices.

- After specifying PdfExportOptions.PasswordSecurityOptions, use the PdfPasswordSecurityOptions.EncryptionLevel property to select the algorithm for encrypting content of a PDF file.

This property returns a PdfEncryptionLevel enumeration value enabling you to select any of the following encryption algorithms:

- **128-bit AES** (Advanced Encryption Standard) - the default value;
- **256-bit AES** (Advanced Encryption Standard);

- **128-bit ARC4** (Alleged Rivest Cipher **4**).
- Use the PdfExportOptions.SignatureOptions to apply an **X.509** certificate to the resulting PDF file and digitally sign the document.

For a code sample, see [How to use the digital signature options when exporting a report to PDF](#).

File Size Optimization

Use the following options to vary the quality of PDF-embedded images and in this way, control the resultant file size.

- PdfExportOptions.ImageQuality
- PdfExportOptions.ConvertImagesToJpeg
When this option is enabled, you can use the PageByPageExportOptionsBase.RasterizationResolution property to define the image resolution.

To reduce the size of large documents (e.g., before sending them via e-mail), you can use the following option.

- PdfExportOptions.NeverEmbeddedFonts

As for the embedded fonts and PDF page content, they are always compressed in the resulting PDF file.

Current Limitations

At present, the following limitations apply to PDF export:

- Glyph Shaping does not work for non-embedded fonts.
- Support for right-to-left languages with non-embedded fonts requires that your application runs under full trust.
- Export of vector EMF and WMF images is not supported. The current export mechanism supports EMF+ only.

See Also

[How To: Convert a DOCX Document to PDF format](#)

[How To: Convert an HTML Document to PDF format](#)

Document Protection

[Office File API](#) > [Word Processing Document API](#) > [Document Protection](#)

Protect the Document from Editing

You can keep anyone from making changes in a document by enabling the document protection. Call the Document.Protect method to protect a document from editing using a given password and protection type (read-only or allow comments only). The default protection type is DocumentProtectionType.ReadOnly. The Document.IsDocumentProtected property indicates whether the document is protected.

Use the Document.Unprotect method to disable protection. Note that it unlocks the document without prompting the user for a password. To prompt the user with a password, execute the UnprotectDocumentCommand command.

Note

The RichEditDocumentServer protection affects only the document modification. The RichEditDocumentServer protection protects the document only against modifications. The document is opened without prompting for a password.

Grant Permission to Users

You can allow certain users to edit parts of a protected document by creating **Range Permissions** as described in the steps below.

1. Define the users and user group lists. By default, the users list is empty and the user group list contains predefined entries (**Everyone**, **Administrators**, **Contributors**, **Owners**, **Editors** and **Current User**). Usernames and group names are independent because the RichEditDocumentServer does not associate users with groups. ☐

The edit permission is given depending on the authentication credentials. The edit permission depends on a user's credentials. Use the AuthenticationOptions class properties to define the current user's identity.

C#

//Define the user credentials:
richEditDocumentServer.Options.Authentication.UserName = "Nancy Skywalker";
richEditDocumentServer.Options.Authentication.Group = "Skywalkers";

Visual Basic

'Define the user credentials:
richEditDocumentServer.Options.Authentication.UserName = "Nancy Skywalker"
richEditDocumentServer.Options.Authentication.Group = "Skywalkers"

2. Call the SubDocument.BeginUpdateRangePermissions method to access the document range permissions collection.
3. Create a new RangePermission object using the RangePermissionCollection.CreateRangePermission method.
4. Utilize the RangePermission.UserName or RangePermission.Group property to specify the specific user or group of users that are allowed to edit the document range.

The table below shows what properties should have the same value to make the range editable to the current user.

This property value	Should be equal to
AuthenticationOptions.Group	RangePermission.Group "Everyone"
AuthenticationOptions.UserName AuthenticationOptions.EMail	RangePermission.UserName

5. Add the created object to the RangePermissionCollection.
6. Finalize the modification by calling the SubDocument.EndUpdateRangePermissions method.
7. Enable document protection. When the protection is enabled, ranges without editing permissions are read-only.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ProtectionActions.cs)
server.LoadDocument("Documents//Grimm.docx", DocumentFormat.OpenXml);
Document document = server.Document;
// Protect document range
RangePermissionCollection rangePermissions = document.BeginUpdateRangePermissions();
RangePermission rp = rangePermissions.CreateRangePermission(document.Paragraphs[3].Range);
rp.Group = "Everyone";
rangePermissions.Add(rp);
document.EndUpdateRangePermissions(rangePermissions);
// Enforce protection and set password.
document.Protect("123");
```

Visual Basic

```
(ProtectionActions.vb)
server.LoadDocument("Documents//Grimm.docx", DocumentFormat.OpenXml)
Dim document As Document = server.Document
' Protect document range
Dim rangePermissions As RangePermissionCollection = document.BeginUpdateRangePermissions()
Dim rp As RangePermission = rangePermissions.CreateRangePermission(document.Paragraphs(3).Range)
rp.Group = "Everyone"
rangePermissions.Add(rp)
document.EndUpdateRangePermissions(rangePermissions)
' Enforce protection and set password.
document.Protect("123")
```

Floating Objects

[Office File API](#) > [Word Processing Document API](#) > [Floating Objects](#)

The object in a document can be anchored in one of two ways: either *inline*, which means that the object is displayed within the text stream, or *floating*. Floating means that the object is strictly positioned, absolutely or relatively within the document, regardless of the text flow. To specify how text flows around the object, the text wrapping type is used.

To insert a floating picture into the document, use the `ShapeCollection.InsertPicture` method. To insert a floating text box, use the `ShapeCollection.InsertTextBox` method.

A floating object is anchored to a document range (the `Shape.Range` property), but can be positioned anywhere on the page that contains the anchor. When you insert a floating object in a certain document position, the anchor is located at the beginning of the first paragraph that contains the specified position.

You can access and manipulate floating objects in code via the corresponding API. Classes relevant to floating objects are listed in the following table.

Object	Description	Exposed by
Shape	An object that can be placed above and below the text layer of the document, as well as within the document, with text surrounding it.	<code>ShapeCollection.InsertPicture</code> <code>ShapeCollection.InsertTextBox</code> <code>ShapeCollection.Item</code>
<code>ShapeCollection</code>	A collection of shapes in a document.	<code>ReadOnlyDocumentImageCollection.Get</code>
<code>ShapeRelativeHorizontalPosition</code>	Specifies to what the horizontal position of a shape is relative.	<code>Shape.RelativeHorizontalPosition</code>
<code>ShapeRelativeVerticalPosition</code>	Specifies to what the vertical position of a shape is relative.	<code>Shape.RelativeVerticalPosition</code>
<code>ShapeHorizontalAlignment</code>	Specifies the horizontal alignment of a shape.	<code>Shape.HorizontalAlignment</code>
<code>ShapeVerticalAlignment</code>	Specifies the type of vertical alignment to apply.	<code>Shape.VerticalAlignment</code>
<code>ShapeLine</code>	Contains formatting information for the shape's border.	<code>Shape.Line</code>
<code>ShapeFill</code>	Fill formatting for a shape.	<code>Shape.Fill</code>
<code>OfficeImage</code>	Gets a picture residing in the floating object.	<code>Shape.Picture</code>
<code>SubDocument</code>	Provides access to content of the text box.	<code>TextBox.Document</code>

HTML Tag Support

[Office File API](#) > [Word Processing Document API](#) > [HTML Tag Support](#)

- [Supported Tags](#)
- [Unsupported Tags](#)

Supported Tags

The table below lists HTML tags supported by the RichEditDocumentServer. External links are currently processed for inline pictures and style sheets (css files). The ID and Class attributes are interpreted for all tags, including the unlisted ones. These attributes are used to specify a style applied to the document part defined by a certain tag.

Tag	Attribute(s)	Notes
a		
b		
base		
basefont	size color face	
big		
blockquote		
br		
center		
code		
div	page-break-before page-break-after page-break-inside background-color	Only if set to always . Only if set to avoid . Only if set to avoid .
font	size color face	
hr		
h1-h6	align	
head		
html		
i		
img	align scr height width	If not specified, the image is considered as inline.
li	type value	
link	href	

	type media	
meta		
ol	type value align	
script		Text inside this tag is ignored.
small		
span		
strike		
strong		
style		
sub		
sup		
table	align bgcolor border bordercolor cellpadding cellspacing width	
td	align bgcolor bordercolor colspan height nowrap rowspan valign width	Supported in the Internet Explorer only. Sets the TableCellPropertiesBase.NoWrap property. RichEditControl's interpretation of this attribute is different from the HTML browser.
th	any allowed	
tr	align bgcolor bordercolor height valign	Supported in the Internet Explorer only.
title		Text inside this tag is ignored.
u		
ui		

Unsupported Tags:

- <base> tag with *href* attribute;
- <div> tag with *border*, *align* and *float* css attribute;
- tag with *list-style-image* css attribute;
- <margin> tag;

- `<tab>` tag;
- `<table>` tag with *cols* attribute;
- `<td>` tag with *bordercolor* and *nowrap* attributes;
- *!important* declaration;
- *word-wrap* and *break-word* css properties;
- css3 shapes;
- `<ui>` tag with *type* attribute.

Examples

[Office File API](#) > [Word Processing Document API](#) > [Examples](#)

This section provides a full list of examples (grouped by features) contained in this help.

Document Conversion

- [How To: Convert a DOCX Document to HTML format](#)
- [How To: Convert a DOCX Document to PDF format](#)
- [How To: Convert an HMTL Document to PDF format](#)
- [How To: Convert an HTML Document to DOCX format](#)

File Operations

- [How to: Create a New Document](#)
- [How to: Load a Document](#)
- [How to: Save a Document](#)

Text Operations

- [How to: Append Text to the Paragraph](#)
- [How to: Copy and Paste Range](#)
- [How to: Insert Text at the Cursor Position](#)
- [How to: Select Text Programmatically](#)

Formatting

- [How to: Change Formatting of Selected Text](#)
- [How to: Change Formatting of the Current Paragraph](#)
- [How to: Specify Default Document Formatting](#)

Styles

- [How To: Create New Character Style](#)
- [How To: Create New Paragraph Style](#)
- [How To: Create New Linked Style](#)

Lists

- [How to: Create Bulleted List in Code](#)
- [How to: Create Numbered List in Code](#)

Pictures

- [How to: Insert Inline Picture](#)
- [How to: Resize Inline Picture](#)
- [How to: Save Inline Picture to a File](#)
- [How to: Add Floating Picture](#)
- [How to: Position Floating Picture](#)
- [How to: Change Z-Order and Text Wrapping](#)

Text Boxes

- [How to: Insert Text Box](#)
- [How to: Insert Rich Text in the Text Box](#)
- [How to: Distinguish Between Text and Picture Shapes](#)

Tables

- [How to: Insert a Table](#)

- [How to: Create a Table with Fixed Column Width](#)
- [How to: Merge and Split Table Cells](#)
- [How to: Change Table Color](#)
- [How to: Create and Apply Table Style](#)
- [How to: Use Conditional Style](#)
- [How to: Delete a Table](#)

Document Elements

- [How to: Insert Bookmark or Hyperlink](#)
- [How to: Create and Modify Header](#)
- [How to: Create, Edit and Delete Comments](#)
- [How to: Specify Document Properties](#)
- [How to: Insert and Modify Fields](#)
- [How to: Create a Checkbox](#)

Layout

- [How to: Configure the Page Layout Programmatically](#)
- [How to: Create a Three-Column Layout with Uniform Columns](#)
- [How To: Add Line Numbering](#)

Protection

- [How to: Protect a Document](#)
- [How to: Grant Editing Permissions in a Document](#)

Printing

- [How to: Print a Document](#)
- [How to: Print a Document using the PrintableComponentLink](#)
- [How to: Show a Print Preview Form](#)

Export

- [How to: Save Selected Image to a File](#)
- [How to: Determine Formatting Capabilities Required to Export the Document](#)
- [How to: Export Range To Different Formats](#)
- [How to: Retain the Image URI in HTML Document](#)

Files

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Files](#)

This section contains the following examples:

- [How to: Create a New Document](#)
- [How to: Load a Document](#)
- [How to: Save a Document](#)

How to: Create a New Document

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Files](#) > [How to: Create a New Document](#)

The RichEditDocumentServer.CreateNewDocument method is used to create a new document.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(BasicActions.cs)
server.CreateNewDocument();

Visual Basic

(BasicActions.vb)
server.CreateNewDocument();

How to: Load a Document

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Files](#) > [How to: Load a Document](#)

RichEditDocumentServer allows you to load a document from a [file](#), a [data stream](#) or a [string](#) using the RichEditDocumentServer.LoadDocument method or RichEditDocumentServer.RtfText property. Handle related [events](#) to determine a moment when the document can be modified.

Load a Document from a File

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(BasicActions.cs)
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
```

Visual Basic

```
(BasicActions.vb)
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
```

Load a Document from a Stream

C#

```
using System.IO;
using DevExpress.XtraRichEdit;
using DevExpress.XtraRichEdit.API.Native;
using System.Reflection;
// ...
//
// Load document from stream
private void btnLoadStream_Click(object sender, EventArgs e) {
    using(Stream stream = GetResourceStream("MyApplication.Document1.rtf"))
        stream.Seek(0, SeekOrigin.Begin);
    richEditDocumentServer1.LoadDocument(stream, DocumentFormat.Rtf);
    stream.Close();
}
static Stream GetResourceStream(string resourceName) {
    return Assembly.GetExecutingAssembly().GetManifestResourceStream(resou
}
```

Visual Basic

```
Imports System.IO
Imports DevExpress.XtraRichEdit
Imports DevExpress.XtraRichEdit.API.Native
Imports System.Reflection
' ...
' Load document from stream
Private Sub btnLoadStream_Click(ByVal sender As Object, ByVal e As EventArgs
    Handles btnLoadStream.Click
    Using stream As Stream = GetResourceStream("MyApplication.Document1.rtf")
        stream.Seek(0, SeekOrigin.Begin)
        richEditDocumentServer1.LoadDocument(stream, DocumentFormat.Rtf)
        stream.Close()
    End Using
End Sub
Private Shared Function GetResourceStream(ByVal resourceName As String) As Stream
    Return Assembly.GetExecutingAssembly().GetManifestResourceStream(resourceName)
End Function
' Load document from file
Private Sub btnLoadFile_Click(ByVal sender As Object, ByVal e As EventArgs
    Handles btnLoadFile.Click
    richEditControll1.LoadDocument("Document1.rtf", DocumentFormat.Rtf)
End Sub
```

Load a Document from a String

C#
<pre>string rtfString = @"{\rtf1\ansi\ansicpg1252\deff0\deflang1049 {\fonttbl{\f0\fswiss\fprq2\fcharset0 Arial;}} {\f1\fswiss\fcharset0 Arial;}} {\colortbl ;\red0\green0\blue255;} \viewkind4\uc1\pard\cf1\lang1033\b\f0\fs32 Test.\cf0\b0\f1\fs20\par}"; Document document = server.Document; document.RtfText = rtfString;</pre>

Visual Basic
<pre>Dim rtfString As String = "{\rtf1\ansi\ansicpg1252\deff0\deflang1049" & ControlChars.CrLf & _ "{\fonttbl{\f0\fswiss\fprq2\fcharset0 Arial;}} & ControlChars.CrLf & _ "{\f1\fswiss\fcharset0 Arial;}} & ControlChars.CrLf & _ "{\colortbl ;\red0\green0\blue255;}" & ControlChars.CrLf & _ "\viewkind4\uc1\pard\cf1\lang1033\b\f0\fs32 Test.\cf0\b0\f1\fs20\par}" Dim document As Document = server.Document document.RtfText = rtfString</pre>

Related Events

The table below lists events related to the document loading:

Event	Description
RichEditDocumentServer.BeforeImport	Occurs before a document is loaded (imported from an external source).

RichEditDocumentServer.InitializeDocument	Occurs before a document is loaded. Handle this event to set initial document settings.
RichEditDocumentServer.DocumentLoaded	Occurs after a document is loaded.
DocumentLayout.DocumentFormatted	Fires after the document layout is calculated.
RichEditDocumentServer.InvalidFormatException	Fires when the supplied data could not be recognized as data in the assumed format for import.
RichEditDocumentServer.UnhandledException	This event is raised when an exception unhandled by the RichEditDocumentServer occurs.

How to: Save a Document

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Files](#) > [How to: Save a Document](#)

To save a document in the RichEditDocumentServer, use the RichEditDocumentServer.SaveDocument method. The RichEditControlOptionsBase.DocumentSaveOptions property provides access to information about the current and default file names and formats.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(BasicActions.cs)
server.Document.AppendDocumentContent("Documents\\Grimm.docx", DocumentFormat.OpenXml);
server.SaveDocument("SavedDocument.docx", DocumentFormat.OpenXml);
System.Diagnostics.Process.Start("explorer.exe", "/select," + "SavedDocument.docx");
```

Visual Basic

```
(BasicActions.vb)
server.Document.AppendDocumentContent("Documents\\Grimm.docx", DocumentForm
server.SaveDocument("SavedDocument.docx", DocumentFormat.OpenXml)
System.Diagnostics.Process.Start("explorer.exe", "/select," & "SavedDo
```

Note

Consider the situation when a document is being saved to a locked or read-only file. To prevent application failure, subscribe to the RichEditDocumentServer.UnhandledException event and set the RichEditUnhandledExceptionEventArgs.Handled property to **true**.

The RichEditDocumentServer.BeforeExport event is raised before you save a document. You can handle this event to check how the exporter formats a document. For example, the following code snippet determines whether or not a document contains complex formatting elements (inline pictures). If they are found, the user is prompted to save the metafile representation along with the original picture (by enabling the RtfDocumentExporterCompatibilityOptions.DuplicateObjectAsMetafile property). This technique increases the file size, but makes it possible to open this file with third-party editors which don't support native picture format.

C#

```

using DevExpress.XtraRichEdit;
using DevExpress.XtraRichEdit.Export;
private static void Server_BeforeExport(object sender, BeforeExportEventArgs e)
{
    DocumentExportCapabilities checkDocument = server.Document.RequiredExportCapabilities;
    if ((e.DocumentFormat == DocumentFormat.Rtf) && checkDocument.InlineFormat == DocumentFormat.Rtf)
    {
        DialogResult reduceFileSize = MessageBox.Show("This document contains pictures. To save it as a rich text file, you must enable dual picture format. You can embed the same picture in two different types (original and dual picture format) although it increases the file size. By default a picture is saved as dual picture format. Do you want to enable dual picture format in a saved file?",
            "Warning", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
        RtfDocumentExporterOptions options = e.Options as RtfDocumentExporterOptions;
        if (options != null)
        {
            switch (reduceFileSize)
            {
                case DialogResult.Yes:
                    options.Compatibility.DuplicateObjectAsMetafile = true;
                    break;
                case System.Windows.Forms.DialogResult.No:
                    options.Compatibility.DuplicateObjectAsMetafile = false;
                    break;
            }
        }
    }
}

```

Visual Basic

```

Private Sub server_BeforeExport(ByVal sender As Object, ByVal e As DevExpress.XtraRichEdit.Export.BeforeExportEventArgs)
    Dim checkDocument As DocumentExportCapabilities = server.Document.RequiredExportCapabilities
    If (e.DocumentFormat = DocumentFormat.Rtf) AndAlso checkDocument.InlineFormat == DocumentFormat.Rtf Then
        Dim reduceFileSize As DialogResult = MessageBox.Show("This document contains pictures. To save it as a rich text file, you must enable dual picture format. You can embed the same picture in two different types (original and dual picture format) although it increases the file size. By default a picture is saved as dual picture format. Do you want to enable dual picture format in a saved file?",
            "Warning", MessageBoxButtons.YesNo, MessageBoxIcon.Warning)
        Dim options As RtfDocumentExporterOptions = TryCast(e.Options, RtfDocumentExporterOptions)
        If options IsNot Nothing Then
            Select Case reduceFileSize
                Case System.Windows.Forms.DialogResult.Yes
                    options.Compatibility.DuplicateObjectAsMetafile = True
                Case System.Windows.Forms.DialogResult.No
                    options.Compatibility.DuplicateObjectAsMetafile = False
            End Select
        End If
    End If
End Sub

```

Document Conversion

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Conversion](#)

This section contains the following examples:

- [How To: Convert a DOCX Document to PDF format](#)
- [How To: Convert a DOCX Document to HTML format](#)
- [How To: Convert an HTML Document to PDF format](#)
- [How To: Convert an HTML Document to DOCX format](#)

How To: Convert a DOCX Document to PDF format

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Conversion](#) > [How To: Convert a DOCX Document to PDF format](#)

This example demonstrates how to export a document to PDF format using the `PrintingSystemBase.ExportToPdf` method.

To set the export options, an `PdfExportOptions` instance can be used. It provides a number of properties, such as `PdfExportOptions.ImageQuality` and `PdfExportOptions.ConvertImagesToJpeg`, allowing you to adjust the required settings for export.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ExportActions.cs)
server.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml);
//Specify export options:
PdfExportOptions options = new PdfExportOptions();
options.DocumentOptions.Author = "Mark Jones";
options.Compressed = false;
options.ImageQuality = PdfJpegImageQuality.Highest;
//Export the document to the stream:
using (FileStream pdfFileStream = new FileStream("Document_PDF.pdf", FileMode.Create))
{
    server.ExportToPdf(pdfFileStream, options);
}
System.Diagnostics.Process.Start("Document PDF.pdf");
```

Visual Basic

```
(ExportActions.vb)
server.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml)
'Specify export options:
Dim options As New PdfExportOptions()
options.DocumentOptions.Author = "Mark Jones"
options.Compressed = False
options.ImageQuality = PdfJpegImageQuality.Highest
'Export the document to the stream:
Using pdfFileStream As New FileStream("Document_PDF.pdf", FileMode.Create)
    server.ExportToPdf(pdfFileStream, options)
End Using
System.Diagnostics.Process.Start("Document PDF.pdf")
```

How To: Convert a DOCX Document to HTML format

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Conversion](#) > [How To: Convert a DOCX Document to HTML format](#)

This example demonstrates how to export the document to the HTML format using the RichEditDocumentServer.SaveDocument method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ExportActions.cs)
server.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml);
string filePath = "Document_HTML.html";
using (FileStream htmlFileStream = new FileStream(filePath, FileMode.Create))
{
    server.SaveDocument(htmlFileStream, DocumentFormat.Html);
}
System.Diagnostics.Process.Start(filePath);
```

Visual Basic

```
(ExportActions.vb)
server.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml)
Dim filePath As String = "Document_HTML.html"
Using htmlFileStream As New FileStream(filePath, FileMode.Create)
    server.SaveDocument(htmlFileStream, DocumentFormat.Html)
End Using
System.Diagnostics.Process.Start(filePath)
```

How To: Convert an HMTL Document to PDF format

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Conversion](#) > [How To: Convert an HMTL Document to PDF format](#)

The following code sample shows how to export an HTML document to PDF format using the RichEditDocumentServer.ExportToPdf method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(ExportActions.cs)
server.LoadDocument("Documents\\TextWithImages.htm");
server.ExportToPdf("Document_PDF.pdf");
System.Diagnostics.Process.Start("Document PDF.pdf");

Visual Basic

(ExportActions.vb)
server.LoadDocument("Documents\\TextWithImages.htm")
server.ExportToPdf("Document_PDF.pdf")
System.Diagnostics.Process.Start("Document PDF.pdf")

Pass the PdfExportOptions instance to the RichEditDocumentServer.ExportToPdf method to specify the exporting options (image quality, additional metadata, etc.).

Note

Implement the IUriProvider descendant and override the IUriProvider.CreateCssUri or IUriProvider.CreateImageUri method to manage decoding external contents in an HTML file. It executes your method instead of the default one when the RichEditDocumentServer needs to load data from a specific URI. Refer to the [How to: Retain the Image URI in HTML Document](#) topic for details.

How To: Convert an HTML Document to DOCX format

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Conversion](#) > [How To: Convert an HTML Document to DOCX format](#)

The following code example shows how to convert an HTML document to the DOCX format using the RichEditDocumentServer.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(ExportActions.cs)

server.LoadDocument("Documents\\TextWithImages.htm");
server.SaveDocument("Document_DOCX.docx", DocumentFormat.OpenXml);
System.Diagnostics.Process.Start("Document_DOCX.docx");

Visual Basic

(ExportActions.vb)

server.LoadDocument("Documents\\TextWithImages.htm")
server.SaveDocument("Document_DOCX.docx", DocumentFormat.OpenXml)
System.Diagnostics.Process.Start("Document_DOCX.docx")

Note

Implement the IUriProvider descendant and override the IUriProvider.CreateCssUri or IUriProvider.CreateImageUri method to manage decoding external contents in an HTML file. It executes your method instead of the default one when the RichEditDocumentServer needs to load data from a specific URI. Refer to the [How to: Retain the Image URI in HTML Document](#) topic for details.

Text

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text](#)

This section contains the following examples:

- [How to: Select Text Programmatically](#)
- [How to: Insert Text at the Cursor Position](#)
- [How to: Append Text to the Paragraph](#)
- [How to: Copy and Paste Range](#)

How to: Select Text Programmatically

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text](#) > [How to: Select Text Programmatically](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

Use the `Document.Selection` property to select a text. The following code snippet illustrates how you can select a range of 69 positions starting from the position 216 in the document:

C#

```
(RangeActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
DocumentPosition myStart = document.CreatePosition(69);
DocumentRange myRange = document.CreateRange(myStart, 216);
document.Selection = myRange;
```

Visual Basic

```
(RangeActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim myStart As DocumentPosition = document.CreatePosition(69)
Dim myRange As DocumentRange = document.CreateRange(myStart, 216)
document.Selection = myRange
```

How to: Insert Text at the Cursor Position

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text](#) > [How to: Insert Text at the Cursor Position](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

The following code snippet demonstrates how you can insert text at the cursor position in the RichEditControl document:

C#

```
(RangeActions.cs)
Document document = server.Document;
DocumentPosition pos = document.CaretPosition;
SubDocument doc = pos.BeginUpdateDocument();
doc.InsertText(pos, " INSERTED TEXT ");
pos.EndUpdateDocument(doc);
```

Visual Basic

```
(RangeActions.vb)
Dim document As Document = server.Document
Dim pos As DocumentPosition = document.CaretPosition
Dim doc As SubDocument = pos.BeginUpdateDocument()
doc.InsertText(pos, " INSERTED TEXT ")
pos.EndUpdateDocument(doc)
```

How to: Append Text to the Paragraph

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text](#) > [How to: Append Text to the Paragraph](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

The following code snippet appends text to the end of the current paragraph. Note that the `DocumentPosition.BeginUpdateDocument` call is required for proper operation within header/footer since it returns a `SubDocument` interface.

C#

```
(RangeActions.cs)
Document document = server.Document;
document.BeginUpdate();
document.AppendText("First Paragraph\nSecond Paragraph\nThird Paragraph");
document.EndUpdate();
DocumentPosition pos = document.CaretPosition;
SubDocument doc = pos.BeginUpdateDocument();
Paragraph par = doc.Paragraphs.Get(pos);
DocumentPosition newPos = doc.CreatePosition(par.Range.End.ToInt() - 1);
doc.InsertText(newPos, "<<Appended to Paragraph End>>");
pos.EndUpdateDocument(doc);
```

Visual Basic

```
(RangeActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
document.AppendText("First Paragraph" & vbCrLf & "Second Paragraph" & vbCrLf &
document.EndUpdate()
Dim pos As DocumentPosition = document.CaretPosition
Dim doc As SubDocument = pos.BeginUpdateDocument()
Dim par As Paragraph = doc.Paragraphs.Get(pos)
Dim newPos As DocumentPosition = doc.CreatePosition(par.Range.End.ToInt()
doc.InsertText(newPos, "<<Appended to Paragraph End>>")
pos.EndUpdateDocument(doc)
```

How to: Copy and Paste Range

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text](#) > [How to: Copy and Paste Range](#)

The following code snippet demonstrates how to copy and paste range using Document.Copy and Document.Paste methods.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(RangeActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
DocumentRange myRange = document.Paragraphs[0].Range;
document.Copy(myRange);
document.Paste(DocumentFormat.PlainText);
```

Visual Basic

```
(RangeActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim myRange As DocumentRange = document.Paragraphs(0).Range
document.Copy(myRange)
document.Paste(DocumentFormat.PlainText)
```

Formatting

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Formatting](#)

This section contains the following examples:

- [How to: Change Formatting of Selected Text](#)
- [How to: Change Formatting of the Current Paragraph](#)
- [How to: Specify Default Document Formatting](#)

How to: Change Formatting of Selected Text

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Formatting](#) > [How to: Change Formatting of Selected Text](#)

This code sample demonstrates how you can get the selected text in code, and modify its attributes.

To apply direct formatting to characters in a range, call the `SubDocument.BeginUpdateCharacters` method for the specified range, modify the properties of the returned `CharacterProperties` object, and subsequently call the `SubDocument.EndUpdateCharacters` method to finalize the modification.

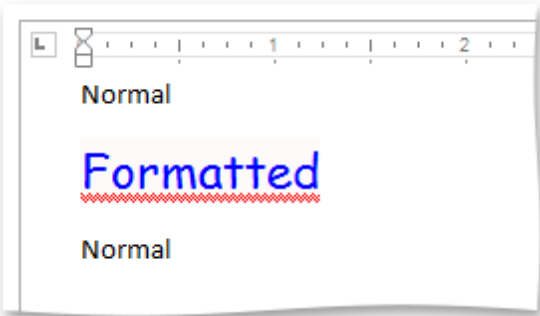
The `Document.Selection` property is used to obtain the `DocumentRange` object, representing the user selection. To change default character formatting for the current document, use the `Document.DefaultCharacterProperties` property.

Note

Make sure that the `SubDocument.BeginUpdateCharacters` method always has its corresponding `SubDocument.EndUpdateCharacters` method to ensure proper operation of the control.

The following code snippet changes the font and the color of the specified paragraph.

The result is shown below:



Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(FormattingActions.cs)
Document document = server.Document;
document.BeginUpdate();
document.AppendText("Normal\nFormatted\nNormal");
document.EndUpdate();
DocumentRange range = document.Paragraphs[1].Range;
CharacterProperties cp = document.BeginUpdateCharacters(range);
cp.FontName = "Comic Sans MS";
cp.FontSize = 18;
cp.ForeColor = Color.Blue;
cp.BackColor = Color.Snow;
cp.Underline = UnderlineType.DoubleWave;
cp.UnderlineColor = Color.Red;
document.EndUpdateCharacters(cp);

Visual Basic

```
(FormattingActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
document.AppendText("Normal" & vbCrLf & "Formatted" & vbCrLf & "Normal")
document.EndUpdate()
Dim range As DocumentRange = document.Paragraphs(1).Range
Dim cp As CharacterProperties = document.BeginUpdateCharacters(range)
cp.FontName = "Comic Sans MS"
cp.FontSize = 18
cp.ForeColor = Color.Blue
cp.BackColor = Color.Snow
cp.Underline = UnderlineType.DoubleWave
cp.UnderlineColor = Color.Red
document.EndUpdateCharacters(cp)
```

How to: Change Formatting of the Current Paragraph

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Formatting](#) > [How to: Change Formatting of the Current Paragraph](#)

This code sample demonstrates how you can modify the attributes of the current paragraph. The Document.Selection property is used to obtain the DocumentRange object, representing the user selection.

To modify paragraph formatting, such as the ParagraphPropertiesBase.LineSpacing, Paragraph.Alignment, Paragraph.SpacingBefore etc., call the SubDocument.BeginUpdateParagraphs method for the specified range, modify the properties of the returned ParagraphProperties object and call the SubDocument.EndUpdateParagraphs method to finalize the modification.

The following code snippet changes the first line indent and the line spacing of the paragraph containing the selection.

Note

To change default paragraph formatting for the current document, use the Document.DefaultParagraphProperties property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(FormattingActions.cs)
Document document = server.Document;
document.BeginUpdate();
document.AppendText("Modified Paragraph\nNormal\nNormal");
document.EndUpdate();
DocumentPosition pos = document.Range.Start;
DocumentRange range = document.CreateRange(pos, 0);
ParagraphProperties pp = document.BeginUpdateParagraphs(range);
// Center paragraph
pp.Alignment = ParagraphAlignment.Center;
// Set triple spacing
pp.LineSpacingType = ParagraphLineSpacing.Multiple;
pp.LineSpacingMultiplier = 3;
// Set left indent at 0.5".
// Default unit is 1/300 of an inch (a document unit).
pp.LeftIndent = DevExpress.Office.Utils.Units.InchesToDocumentsF(0.5f);
// Set tab stop at 1.5"
TabInfoCollection tbiColl = pp.BeginUpdateTabs(true);
TabInfo tbi = new DevExpress.XtraRichEdit.API.Native.TabInfo();
tbi.Alignment = TabAlignmentType.Center;
tbi.Position = DevExpress.Office.Utils.Units.InchesToDocumentsF(1.5f);
tbiColl.Add(tbi);
pp.EndUpdateTabs(tbiColl);
document.EndUpdateParagraphs(pp);

Visual Basic

```
(FormattingActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
document.AppendText("Modified Paragraph" & vbCrLf & "Normal" & vbCrLf & "Normal")
document.EndUpdate()
Dim pos As DocumentPosition = document.Range.Start
Dim range As DocumentRange = document.CreateRange(pos, 0)
Dim pp As ParagraphProperties = document.BeginUpdateParagraphs(range)
' Center paragraph
pp.Alignment = ParagraphAlignment.Center
' Set triple spacing
pp.LineSpacingType = ParagraphLineSpacing.Multiple
pp.LineSpacingMultiplier = 3
' Set left indent at 0.5".
' Default unit is 1/300 of an inch (a document unit).
pp.LeftIndent = DevExpress.Office.Utils.Units.InchesToDocumentsF(0.5F)
' Set tab stop at 1.5"
Dim tbiColl As TabInfoCollection = pp.BeginUpdateTabs(True)
Dim tbi As TabInfo = New DevExpress.XtraRichEdit.API.Native.TabInfo()
tbi.Alignment = TabAlignmentType.Center
tbi.Position = DevExpress.Office.Utils.Units.InchesToDocumentsF(1.5F)
tbiColl.Add(tbi)
pp.EndUpdateTabs(tbiColl)
document.EndUpdateParagraphs(pp)
```

How to: Specify Default Document Formatting

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Formatting](#) > [How to: Specify Default Document Formatting](#)

Use the `Document.DefaultCharacterProperties` and `Document.DefaultParagraphProperties` properties available via the **RichEditDocumentServer.Document.DefaultCharacterProperties** and **RichEditDocumentServer.Document.DefaultParagraphProperties** notations.

Changing default setting will change those portions of the text that have not been formatted explicitly, but leave already formatted text as it is. For example, default font is Arial 12. If I set a selection to Times New Roman 14 and then set the default setting to Arial 10, the text formatted with Times New Roman 14 remains unchanged.

The following code snippet illustrates how to specify default document formatting when a new empty document is created. A new empty document can be created by executing the `CreateEmptyDocumentCommand` command (pressing CTRL-N).

This member supports the internal infrastructure, and is not intended to be used directly from your code.

C#

```
private void server_EmptyDocumentCreated(object sender, EventArgs e) {  
    server.Document.DefaultCharacterProperties.ForeColor = Color.Red;  
    server.Document.DefaultParagraphProperties.Alignment = ParagraphAlignment.Center;  
    server.Document.AppendText("Document created at " + DateTime.Now.ToLongTimeString()); }  
}
```

Visual Basic

```
Private Sub server_EmptyDocumentCreated(ByVal sender As Object, ByVal e As EventArgs) Handles server.EmptyDocumentCreated  
    server.Document.DefaultCharacterProperties.ForeColor = Color.Red  
    server.Document.DefaultParagraphProperties.Alignment = ParagraphAlignment.Center  
    server.Document.AppendText("Document created at " & DateTime.Now.ToLongTimeString())  
End Sub
```

Styles

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Styles](#)

The section contains the following items:

- [How To: Create New Character Style](#)
- [How To: Create New Paragraph Style](#)
- [How To: Create New Linked Style](#)

How To: Create New Character Style

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Styles](#) > [How To: Create New Character Style](#)

The following example illustrates how to create a character style in code.

- To achieve the required result, do the following:
- Create a new style using the `CharacterStyleCollection.CreateNew` method;
 - Specify the style settings with the use of corresponding `CharacterStyle` object properties;
 - Add the style to the collection of character styles, represented by the `CharacterStyleCollection` object. To do that, use the `CharacterStyleCollection.Add` method. The collection is accessible through the `Document.CharacterStyles` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(StylesActions.cs)
Document document = server.Document;
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
CharacterStyle cstyle = document.CharacterStyles["MyCStyle"];
if (cstyle == null)
{
    cstyle = document.CharacterStyles.CreateNew();
    cstyle.Name = "MyCStyle";
    cstyle.Parent = document.CharacterStyles["Default Paragraph Font"];
    cstyle.ForeColor = System.Drawing.Color.DarkOrange;
    cstyle.Strikeout = StrikeoutType.Double;
    cstyle.FontName = "Verdana";
    document.CharacterStyles.Add(cstyle);
}
DocumentRange myRange = document.Paragraphs[0].Range;
CharacterProperties charProps =
    document.BeginUpdateCharacters(myRange);
charProps.Style = cstyle;
document.EndUpdateCharacters(charProps);
```

Visual Basic

```
(StylesActions.vb)
Dim document As Document = server.Document
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim cstyle As CharacterStyle = document.CharacterStyles("MyCStyle")
If cstyle Is Nothing Then
    cstyle = document.CharacterStyles.CreateNew()
    cstyle.Name = "MyCStyle"
    cstyle.Parent = document.CharacterStyles("Default Paragraph Font")
    cstyle.ForeColor = System.Drawing.Color.DarkOrange
    cstyle.Strikeout = StrikeoutType.Double
    cstyle.FontName = "Verdana"
    document.CharacterStyles.Add(cstyle)
End If
Dim myRange As DocumentRange = document.Paragraphs(0).Range
Dim charProps As CharacterProperties = document.BeginUpdateCharacters(myRange)
charProps.Style = cstyle
document.EndUpdateCharacters(charProps)
```

How To: Create New Paragraph Style

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Styles](#) > [How To: Create New Paragraph Style](#)

The following example shows how to create a paragraph style.

To create a paragraph style, perform the following steps:

- Create a new paragraph style using ParagraphStyleCollection.CreateNew method;
- Specify the required style settings using corresponding ParagraphStyle instance properties;
- Add it to the ParagraphStyleCollection, accessible through the Document.ParagraphStyles property. To do that, use the ParagraphStyleCollection.Add method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(StylesActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DevExpress.XtraRichEdit.DocumentFormat.OpenXml);
ParagraphStyle pstyle = document.ParagraphStyles["MyPStyle"];
if (pstyle == null)
{
    pstyle = document.ParagraphStyles.CreateNew();
    pstyle.Name = "MyPStyle";
    pstyle.LineSpacingType = ParagraphLineSpacing.Double;
    pstyle.Alignment = ParagraphAlignment.Center;
    document.ParagraphStyles.Add(pstyle);
}
document.Paragraphs[2].Style = pstyle;
```

Visual Basic

```
(StylesActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DevExpress.XtraRichEdit.Docu
Dim pstyle As ParagraphStyle = document.ParagraphStyles("MyPStyle")
If pstyle Is Nothing Then
    pstyle = document.ParagraphStyles.CreateNew()
    pstyle.Name = "MyPStyle"
    pstyle.LineSpacingType = ParagraphLineSpacing.Double
    pstyle.Alignment = ParagraphAlignment.Center
    document.ParagraphStyles.Add(pstyle)
End If
document.Paragraphs(2).Style = pstyle
```

How To: Create New Linked Style

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Styles](#) > [How To: Create New Linked Style](#)

This example explains how to create a linked style ☐ a style that can be applied both to the character and paragraph, depending on the range to which it is applied.

To create such a style, perform the following steps:

1. Create new paragraph style;

2. Create new character style;

3. Set the CharacterStyle.LinkedStyle to the newly created paragraph style, or set the ParagraphStyle.LinkedStyle property to the corresponding CharacterStyle object. It is sufficient to set only one of these properties.

Note

All the actions should be performed using the SubDocument.BeginUpdate - SubDocument.EndUpdate paired methods. Otherwise, flickering might occur when starting the application.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(StylesActions.cs)
Document document = server.Document;
document.BeginUpdate();
document.AppendText("Line One\nLine Two\nLine Three");
document.EndUpdate();
ParagraphStyle lstyle = document.ParagraphStyles["MyLinkedStyle"];
if (lstyle == null)
{
 document.BeginUpdate();
 lstyle = document.ParagraphStyles.CreateNew();
 lstyle.Name = "MyLinkedStyle";
 lstyle.LineSpacingType = ParagraphLineSpacing.Double;
 lstyle.Alignment = ParagraphAlignment.Center;
 document.ParagraphStyles.Add(lstyle);
 CharacterStyle lcstyle = document.CharacterStyles.CreateNew();
 lcstyle.Name = "MyLinkedCStyle";
 document.CharacterStyles.Add(lcstyle);
 lcstyle.LinkedStyle = lstyle;
 lcstyle.ForeColor = System.Drawing.Color.DarkGreen;
 lcstyle.Strikeout = StrikeoutType.Single;
 lcstyle.FontSize = 24;
 document.EndUpdate();
 document.SaveDocument("LinkedStyleSample.docx", DevExpress.XtraRichEdit.DocumentFormat.OpenXml);
 System.Diagnostics.Process.Start("explorer.exe", "/select," + "LinkedStyleSample.docx");
}

Visual Basic

```
(StylesActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
document.AppendText("Line One" & vbCrLf & "Line Two" & vbCrLf & "Line Three")
document.EndUpdate()
Dim lstyle As ParagraphStyle = document.ParagraphStyles("MyLinkedStyle")
If lstyle Is Nothing Then
    document.BeginUpdate()
    lstyle = document.ParagraphStyles.CreateNew()
    lstyle.Name = "MyLinkedStyle"
    lstyle.LineSpacingType = ParagraphLineSpacing.Double
    lstyle.Alignment = ParagraphAlignment.Center
    document.ParagraphStyles.Add(lstyle)
    Dim lcstyle As CharacterStyle = document.CharacterStyles.CreateNew()
    lcstyle.Name = "MyLinkedCStyle"
    document.CharacterStyles.Add(lcstyle)
    lcstyle.LinkedStyle = lstyle
    lcstyle.ForeColor = System.Drawing.Color.DarkGreen
    lcstyle.Strikeout = StrikeoutType.Single
    lcstyle.FontSize = 24
    document.EndUpdate()
    document.SaveDocument("LinkedStyleSample.docx", DevExpress.XtraRichEdit
System.Diagnostics.Process.Start("explorer.exe", "/select," & "LinkedS
End If
```

Lists

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Lists](#)

This section contains the following examples:

- [How to: Create Bulleted List in Code](#)
- [How to: Create Numbered List in Code](#)

How to: Create Bulleted List in Code

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Lists](#) > [How to: Create Bulleted List in Code](#)

This example illustrates how to create a standard bulleted list in code.

1. Add a new AbstractNumberingList instance to the Document.AbstractNumberingLists collection
2. Specify a NumberingListBase.NumberingType property, so that it equals the NumberingFormat.Bullet value.
3. For each list level, specify the paragraph characteristics using the ListLevel.ParagraphProperties property. Set the left indentation (the ParagraphPropertiesBase.LeftIndent property) so that each level has a different offset from the left. To specify a symbol that will be used to mark a list item, set the font via the ListLevel.CharacterProperties property and the character via the ListLevelProperties.DisplayFormatString property.
4. Use the **Add** method of the collection of lists in the document (accessible via the Document.NumberingLists property) to add a list definition to the document. The parameter is the index of an abstract numbering list created previously. The **Add** method creates a pattern that can be applied to a paragraph so that the paragraph looks like a list item.
5. To include a paragraph in a bulleted list, set the Paragraph.ListIndex property of a paragraph to the index of a list in the document and specify the Paragraph.ListLevel property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ListsActions.cs)
Document document = server.Document;
document.BeginUpdate();
// Define an abstract list that is the pattern for lists used in the document.
AbstractNumberingList list = document.AbstractNumberingLists.Add();
list.NumberingType = NumberingType.Bullet;
// Specify parameters for each list level.
ListLevel level = list.Levels[0];
level.ParagraphProperties.LeftIndent = 100;
level.CharacterProperties.FontName = "Symbol";
level.DisplayFormatString = new string('\u00B7', 1);
level = list.Levels[1];
level.ParagraphProperties.LeftIndent = 250;
level.CharacterProperties.FontName = "Symbol";
level.DisplayFormatString = new string('\u006F', 1);
level = list.Levels[2];
level.ParagraphProperties.LeftIndent = 450;
level.CharacterProperties.FontName = "Symbol";
level.DisplayFormatString = new string('\u00B7', 1);
// Create a list for use in the document. It is based on a previously defined abstract list with ID = 0.
document.NumberingLists.Add(0);
document.EndUpdate();
document.AppendText("Line 1\nLine 2\nLine 3");
// Convert all paragraphs to list items.
document.BeginUpdate();
ParagraphCollection paragraphs = document.Paragraphs;
foreach (Paragraph pgf in paragraphs)
{
    pgf.ListIndex = 0;
    pgf.ListLevel = 1;
}
document.EndUpdate();
```

Visual Basic

```
(ListsActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
' Define an abstract list that is the pattern for lists used in the document
Dim list As AbstractNumberingList = document.AbstractNumberingLists.Add()
list.NumberingType = NumberingType.Bullet
' Specify parameters for each list level.
Dim level As ListLevel = list.Levels(0)
level.ParagraphProperties.LeftIndent = 100
level.CharacterProperties.FontName = "Symbol"
level.DisplayFormatString = New String(ChrW(&H00B7), 1)
level = list.Levels(1)
level.ParagraphProperties.LeftIndent = 250
level.CharacterProperties.FontName = "Symbol"
level.DisplayFormatString = New String(ChrW(&H006F), 1)
level = list.Levels(2)
level.ParagraphProperties.LeftIndent = 450
level.CharacterProperties.FontName = "Symbol"
level.DisplayFormatString = New String(ChrW(&H00B7), 1)
' Create a list for use in the document. It is based on a previously defined list.
document.NumberingLists.Add(0)
document.EndUpdate()
document.AppendText("Line 1" & vbCrLf & "Line 2" & vbCrLf & "Line 3")
' Convert all paragraphs to list items.
document.BeginUpdate()
Dim paragraphs As ParagraphCollection = document.Paragraphs
For Each pgf As Paragraph In paragraphs
    pgf.ListIndex = 0
    pgf.ListLevel = 1
Next pgf
document.EndUpdate()
```

How to: Create Numbered List in Code

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Lists](#) > [How to: Create Numbered List in Code](#)

To create a numbered list in code, perform the following steps.

1. Add a new `AbstractNumberingList` instance to the `Document.AbstractNumberingLists` collection.
2. Specify a `NumberingListBase.NumberingType` property, so that it is equal to the `NumberingType.MultiLevel` value.
3. For each list level, specify the paragraph characteristics using the `ListLevel.ParagraphProperties` property. Set the left indentation so that each level has a different offset from the left. Specify a hanging first line and set the first line indent value to provide enough space for a number that precedes the text.
4. Set the `ListLevelProperties.Start` value that is the number from which to start the list.
5. Specify a format used to display a number via the `ListLevelProperties.DisplayFormatString` property.
6. Use the **Add** method of the collection of lists in the document (accessible via the `Document.NumberingLists` property) to add a list definition to the document. The parameter is the index of an abstract numbering list created previously. The **Add** method creates a pattern that can be applied to a paragraph so that the paragraph looks like a list item.
7. To include a paragraph in a list, set the `Paragraph.ListIndex` property of a paragraph to the index of a list in the document and specify the `Paragraph.ListLevel` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ListsActions.cs)
Document document = server.Document;
document.BeginUpdate();
// Define an abstract list that is the pattern for lists used in the document.
AbstractNumberingList list = document.AbstractNumberingLists.Add();
list.NumberingType = NumberingType.MultiLevel;
// Specify parameters for each list level.
ListLevel level = list.Levels[0];
level.ParagraphProperties.LeftIndent = 150;
level.ParagraphProperties.FirstLineIndentType = ParagraphFirstLineIndent.Hanging;
level.ParagraphProperties.FirstLineIndent = 75;
level.Start = 1;
level.NumberingFormat = NumberingFormat.Decimal;
level.DisplayFormatString = "{0}";
level = list.Levels[1];
level.ParagraphProperties.LeftIndent = 300;
level.ParagraphProperties.FirstLineIndentType = ParagraphFirstLineIndent.Hanging;
level.ParagraphProperties.FirstLineIndent = 150;
level.Start = 1;
level.NumberingFormat = NumberingFormat.DecimalEnclosedParentheses;
level.DisplayFormatString = "{0}?{1}";
level = list.Levels[2];
level.ParagraphProperties.LeftIndent = 450;
level.ParagraphProperties.FirstLineIndentType = ParagraphFirstLineIndent.Hanging;
level.ParagraphProperties.FirstLineIndent = 220;
level.Start = 1;
level.NumberingFormat = NumberingFormat.LowerRoman;
level.DisplayFormatString = "{0}?{1}?{2}";
// Create a list for use in the document. It is based on a previously defined abstract list with ID = 0.
document.NumberingLists.Add(0);
document.EndUpdate();
document.AppendText("Line one\nLine two\nLine three\nLine four");
// Convert all paragraphs to list items of level 0.
document.BeginUpdate();
ParagraphCollection paragraphs = document.Paragraphs;
foreach (Paragraph pgf in paragraphs)
{
    pgf.ListIndex = 0;
    pgf.ListLevel = 0;
}
// Specify a different level for a certain paragraph.
document.Paragraphs[1].ListLevel = 1;
document.EndUpdate();
```

Visual Basic

```
(ListsActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
' Define an abstract list that is the pattern for lists used in the document
Dim list As AbstractNumberingList = document.AbstractNumberingLists.Add()
list.NumberingType = NumberingType.MultiLevel
' Specify parameters for each list level.
Dim level As ListLevel = list.Levels(0)
level.ParagraphProperties.LeftIndent = 150
level.ParagraphProperties.FirstLineIndentType = ParagraphFirstLineIndent.H
level.ParagraphProperties.FirstLineIndent = 75
level.Start = 1
level.NumberingFormat = NumberingFormat.Decimal
level.DisplayFormatString = "{0}"
level = list.Levels(1)
level.ParagraphProperties.LeftIndent = 300
level.ParagraphProperties.FirstLineIndentType = ParagraphFirstLineIndent.H
level.ParagraphProperties.FirstLineIndent = 150
level.Start = 1
level.NumberingFormat = NumberingFormat.DecimalEnclosedParentheses
level.DisplayFormatString = "{0}?{1}"
level = list.Levels(2)
level.ParagraphProperties.LeftIndent = 450
level.ParagraphProperties.FirstLineIndentType = ParagraphFirstLineIndent.H
level.ParagraphProperties.FirstLineIndent = 220
level.Start = 1
level.NumberingFormat = NumberingFormat.LowerRoman
level.DisplayFormatString = "{0}?{1}?{2}"
' Create a list for use in the document. It is based on a previously defined list.
document.NumberingLists.Add(0)
document.EndUpdate()
document.AppendText("Line one" & vbCrLf & "Line two" & vbCrLf & "Line three" & vbCrLf)
' Convert all paragraphs to list items of level 0.
document.BeginUpdate()
Dim paragraphs As ParagraphCollection = document.Paragraphs
For Each pgf As Paragraph In paragraphs
    pgf.ListIndex = 0
    pgf.ListLevel = 0
Next pgf
' Specify a different level for a certain paragraph.
document.Paragraphs(1).ListLevel = 1
document.EndUpdate()
```

Pictures

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Pictures](#)

This section contains the following examples:

- [How to: Insert Inline Picture](#)
- [How to: Resize Inline Picture](#)
- [How to: Save Inline Picture to a File](#)
- [How to: Add Floating Picture](#)
- [How to: Position Floating Picture](#)
- [How to: Change Z-Order and Text Wrapping](#)

How to: Insert Inline Picture

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Pictures](#) > [How to: Insert Inline Picture](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

To insert a picture in line with text, use the `DocumentImageCollection.Insert` method.

C#

(InlinePictureActions.cs)
Document document = server.Document;
DocumentPosition pos = document.Range.Start;
document.Images.Insert(pos, DocumentImageSource.FromFile("Documents\\beverages.png"));

Visual Basic

(InlinePictureActions.vb)
Dim document As Document = server.Document
Dim pos As DocumentPosition = document.Range.Start
document.Images.Insert(pos, DocumentImageSource.FromFile("Documents\\bevera

How to: Resize Inline Picture

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Pictures](#) > [How to: Resize Inline Picture](#)

To change the size of the inline picture, get the collection of inline pictures using the `SubDocument.Images` property and change the scaling factor of the selected image using the `DocumentImage.ScaleX` and `DocumentImage.ScaleY` properties. You can also specify the image size directly using the `DocumentImage.Size` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

The following example examines all images in the document. If the width of an image exceeds 50 millimeters, the image is scaled proportionally to half its size.

C#

```
(InlinePictureActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
ReadOnlyDocumentImageCollection images = document.Images;
// If the width of an image exceeds 50 millimeters,
// the image is scaled proportionally to half its size.
for (int i = 0; i < images.Count; i++)
{
    if (images[i].Size.Width > DevExpress.Office.Utils.Units.MillimetersToDocumentsF(50))
    {
        images[i].ScaleX /= 2;
        images[i].ScaleY /= 2;
    }
}
```

Visual Basic

```
(InlinePictureActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim images As ReadOnlyDocumentImageCollection = document.Images
' If the width of an image exceeds 50 millimeters,
' the image is scaled proportionally to half its size.
For i As Integer = 0 To images.Count - 1
    If images(i).Size.Width > DevExpress.Office.Utils.Units.MillimetersToDocumentsF(50) Then
        images(i).ScaleX /= 2
        images(i).ScaleY /= 2
    End If
Next i
```

How to: Save Inline Picture to a File

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Pictures](#) > [How to: Save Inline Picture to a File](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

To save an image from the document to a file, use the `ReadOnlyDocumentImageCollection.Get` method to obtain a collection of images within the specified range, and use the `OfficeImage.NativeImage` property, which allows access to a native **System.Drawing.Image** object. This object has the **Save** method that accomplishes the task.

C#

```
(InlinePictureActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml);
DocumentRange myRange = document.CreateRange(0, 100);
ReadOnlyDocumentImageCollection images = document.Images.Get(myRange);
if (images.Count > 0)
{
    DevExpress.Office.Utils.OfficeImage myImage = images[0].Image;
    System.Drawing.Image image = myImage.NativeImage;
    string imageName = String.Format("Image_at_pos_{0}.png", images[0].Range.Start.ToInt());
    image.Save(imageName);
    System.Diagnostics.Process.Start("explorer.exe", "/select," + imageName);
}
```

Visual Basic

```
(InlinePictureActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml)
Dim myRange As DocumentRange = document.CreateRange(0, 100)
Dim images As ReadOnlyDocumentImageCollection = document.Images.Get(myRange)
If images.Count > 0 Then
    Dim myImage As DevExpress.Office.Utils.OfficeImage = images(0).Image
    Dim image As System.Drawing.Image = myImage.NativeImage
    Dim imageName As String = String.Format("Image_at_pos_{0}.png", images(0).Range.Start.ToInt())
    image.Save(imageName)
    System.Diagnostics.Process.Start("explorer.exe", "/select," & imageName)
End If
```

How to: Add Floating Picture

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Pictures](#) > [How to: Add Floating Picture](#)

To add a floating picture to the document, use the ShapeCollection.InsertPicture method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

To add a floating picture to the document, use the ShapeCollection.InsertPicture method.

C#

(ShapesActions.cs)
Document document = server.Document;
document.AppendText("Line One\nLine Two\nLine Three");
Shape myPicture = document.Shapes.InsertPicture(document.CreatePosition(15),
 System.Drawing.Image.FromFile("Documents\\beverages.png"));
myPicture.HorizontalAlignment = ShapeHorizontalAlignment.Center;

Visual Basic

(ShapesActions.vb)
Dim document As Document = server.Document
document.AppendText("Line One" & vbCrLf & "Line Two" & vbCrLf & "Line Three")
Dim myPicture As Shape = document.Shapes.InsertPicture(document.CreatePosi
myPicture.HorizontalAlignment = ShapeHorizontalAlignment.Center

How to: Position Floating Picture

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Pictures](#) > [How to: Position Floating Picture](#)

The following code snippet positions a floating picture by specifying the Shape.Offset from the top and left margins. To allow absolute positioning, the Shape.HorizontalAlignment and Shape.VerticalAlignment properties should be set to **None**.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

This example moves a floating object to a position where its upper left corner is located at 4.5 cm to the right of the left margin and 2 cm below the top margin.

C#

(ShapesActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
document.Unit = DevExpress.Office.DocumentUnit.Centimeter;
Shape myPicture = document.Shapes[1];
// Clear the qualitative positioning to allow positioning by specifying the numerical offset.
myPicture.HorizontalAlignment = ShapeHorizontalAlignment.None;
myPicture.VerticalAlignment = ShapeVerticalAlignment.None;
// Specify the reference item for positioning.
myPicture.RelativeHorizontalPosition = ShapeRelativeHorizontalPosition.LeftMargin;
myPicture.RelativeVerticalPosition = ShapeRelativeVerticalPosition.TopMargin;
// Specify the offset value.
myPicture.Offset = new System.Drawing.PointF(4.5f, 2.0f);

Visual Basic

(ShapesActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\Grimm.docx", DocumentFormat.OpenXml)
document.Unit = DevExpress.Office.DocumentUnit.Centimeter
Dim myPicture As Shape = document.Shapes(1)
' Clear the qualitative positioning to allow positioning by specifying the
myPicture.HorizontalAlignment = ShapeHorizontalAlignment.None
myPicture.VerticalAlignment = ShapeVerticalAlignment.None
' Specify the reference item for positioning.
myPicture.RelativeHorizontalPosition = ShapeRelativeHorizontalPosition.LeftMargin
myPicture.RelativeVerticalPosition = ShapeRelativeVerticalPosition.TopMargin
' Specify the offset value.
myPicture.Offset = New System.Drawing.PointF(4.5F, 2.0F)

How to: Change Z-Order and Text Wrapping

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Pictures](#) > [How to: Change Z-Order and Text Wrapping](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

The following code snippet uses the Shape.ZOrder property to position an image behind the specified floating object and sets the Shape.TextWrapping property to set the image behind the text of the document.

C#

(ShapesActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
Shape myPicture = document.Shapes[1];
myPicture.VerticalAlignment = ShapeVerticalAlignment.Top;
myPicture.ZOrder = document.Shapes[0].ZOrder - 1;
myPicture.TextWrapping = TextWrappingType.BehindText;

Visual Basic

(ShapesActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim myPicture As Shape = document.Shapes(1)
myPicture.VerticalAlignment = ShapeVerticalAlignment.Top
myPicture.ZOrder = document.Shapes(0).ZOrder - 1
myPicture.TextWrapping = TextWrappingType.BehindText

Text Boxes

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text Boxes](#)

This section contains the following examples:

- [How to: Insert Text Box](#)
- [How to: Insert Rich Text in the Text Box](#)
- [How to: Distinguish Between Text and Picture Shapes](#)

How to: Insert Text Box

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text Boxes](#) > [How to: Insert Text Box](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ShapesActions.cs)
Document document = server.Document;
document.AppendText("Line One\nLine Two\nLine Three");
Shape myTextBox = document.Shapes.InsertTextBox(document.CreatePosition(15));
myTextBox.HorizontalAlignment = ShapeHorizontalAlignment.Center;
// Specify the text box background color.
myTextBox.Fill.Color = System.Drawing.Color.WhiteSmoke;
// Draw a border around the text box.
myTextBox.Line.Color = System.Drawing.Color.Black;
myTextBox.Line.Thickness = 1;
// Modify text box content.
SubDocument textBoxDocument = myTextBox.TextBox.Document;
textBoxDocument.AppendText("TextBox Text");
CharacterProperties cp = textBoxDocument.BeginUpdateCharacters(textBoxDocument.Range.Start, 7);
cp.ForeColor = System.Drawing.Color.Orange;
cp.FontSize = 24;
textBoxDocument.EndUpdateCharacters(cp);
```

Visual Basic

```
(ShapesActions.vb)
Dim document As Document = server.Document
document.AppendText("Line One" & vbCrLf & "Line Two" & vbCrLf & "Line Three")
Dim myTextBox As Shape = document.Shapes.InsertTextBox(document.CreatePosition(15))
myTextBox.HorizontalAlignment = ShapeHorizontalAlignment.Center
' Specify the text box background color.
myTextBox.Fill.Color = System.Drawing.Color.WhiteSmoke
' Draw a border around the text box.
myTextBox.Line.Color = System.Drawing.Color.Black
myTextBox.Line.Thickness = 1
' Modify text box content.
Dim textBoxDocument As SubDocument = myTextBox.TextBox.Document
textBoxDocument.AppendText("TextBox Text")
Dim cp As CharacterProperties = textBoxDocument.BeginUpdateCharacters(textBoxDocument.Range.Start, 7)
cp.ForeColor = System.Drawing.Color.Orange
cp.FontSize = 24
textBoxDocument.EndUpdateCharacters(cp)
```

How to: Insert Rich Text in the Text Box

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text Boxes](#) > [How to: Insert Rich Text in the Text Box](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

The following code inserts a paragraph and a picture from the main document into the text box. The `TextBox.Document` property provides access to the text box content.

C#

```
(ShapesActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
Shape myTextBox = document.Shapes[0];
// Allow text box resize to fit contents.
myTextBox.TextBox.HeightRule = TextBoxSizeRule.Auto;
SubDocument boxedDocument = myTextBox.TextBox.Document;
int appendPosition = myTextBox.TextBox.Document.Range.End.ToInt();
// Append the second paragraph of the main document to the boxed text.
DocumentRange newRange = boxedDocument.AppendDocumentContent(document.Paragraphs[1].Range);
boxedDocument.Paragraphs.Insert(newRange.Start);
// Insert an image form the main document into the text box.
boxedDocument.Images.Insert(boxedDocument.CreatePosition(appendPosition), document.Images[0].Image.NativeImage);
// Resize the image so that its size equals the image in the main document.
boxedDocument.Images[0].Size = document.Images[0].Size;
```

Visual Basic

```
(ShapesActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim myTextBox As Shape = document.Shapes(0)
' Allow text box resize to fit contents.
myTextBox.TextBox.HeightRule = TextBoxSizeRule.Auto
Dim boxedDocument As SubDocument = myTextBox.TextBox.Document
Dim appendPosition As Integer = myTextBox.TextBox.Document.Range.End.ToInt
' Append the second paragraph of the main document to the boxed text.
Dim newRange As DocumentRange = boxedDocument.AppendDocumentContent(document.Paragraphs[1].Range)
boxedDocument.Paragraphs.Insert(newRange.Start)
' Insert an image form the main document into the text box.
boxedDocument.Images.Insert(boxedDocument.CreatePosition(appendPosition), document.Images(0).Image.NativeImage)
' Resize the image so that its size equals the image in the main document.
boxedDocument.Images(0).Size = document.Images(0).Size
```

How to: Distinguish Between Text and Picture Shapes

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Text Boxes](#) > [How to: Distinguish Between Text and Picture Shapes](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

To distinguish between a text box and a floating picture in code, examine the `Shape.TextBox` and `Shape.Picture` properties. If the **Shape.TextBox** is not *null*, the shape is the text box.

The following code rotates text boxes and resizes floating pictures.

C#

```
(ShapesActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
foreach (Shape s in document.Shapes)
{
    // Rotate a text box and resize a floating picture.
    if (s.TextBox == null)
    {
        s.ScaleX = 0.1f;
        s.ScaleY = 0.1f;
    }
    else
    {
        s.RotationAngle = 45;
    }
}
```

Visual Basic

```
(ShapesActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\Grimm.docx", DocumentFormat.OpenXml)
For Each s As Shape In document.Shapes
    ' Rotate a text box and resize a floating picture.
    If s.TextBox Is Nothing Then
        s.ScaleX = 0.1F
        s.ScaleY = 0.1F
    Else
        s.RotationAngle = 45
    End If
Next s
```

Tables

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Tables](#)

This section contains the following examples:

- [How to: Insert a Table](#)
- [How to: Create a Table with Fixed Column Width](#)
- [How to: Merge and Split Table Cells](#)
- [How to: Change Table Color](#)
- [How to: Create and Apply Table Style](#)
- [How to: Use Conditional Style](#)
- [How to: Delete a Table](#)

How to: Insert a Table

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Tables](#) > [How to: Insert a Table](#)

The following example illustrates how to insert a table to the document.

To create a new table, use the `TableCollection.Create` method. This method creates a table and adds it to the `TableCollection`, accessible through the `SubDocument.Tables` property.

To insert data to the table, use the `SubDocument.InsertText` or `SubDocument.InsertSingleLineText` method. Use the table cell starting position as the method's argument.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(TablesActions.cs)
Document document = server.Document;
// Insert new table.
Table tbl = document.Tables.Create(document.Range.Start, 1, 3, AutoFitBehaviorType.AutoFitToWindow);
// Create a table header.
document.InsertText(tbl[0, 0].Range.Start, "Name");
document.InsertText(tbl[0, 1].Range.Start, "Size");
document.InsertText(tbl[0, 2].Range.Start, "DateTime");
// Insert table data.
DirectoryInfo dirinfo = new DirectoryInfo("C:\\");
try
{
    tbl.BeginUpdate();
    foreach (FileInfo fi in dirinfo.GetFiles())
    {
        TableRow row = tbl.Rows.Append();
        TableCell cell = row.FirstCell;
        string fileName = fi.Name;
        string fileLength = String.Format("{0:N0}", fi.Length);
        string fileLastTime = String.Format("{0:g}", fi.LastWriteTime);
        document.InsertSingleLineText(cell.Range.Start, fileName);
        document.InsertSingleLineText(cell.Next.Range.Start, fileLength);
        document.InsertSingleLineText(cell.Next.Next.Range.Start, fileLastTime);
    }
    // Center the table header.
    foreach (Paragraph p in document.Paragraphs.Get(tbl.FirstRow.Range))
    {
        p.Alignment = ParagraphAlignment.Center;
    }
}
finally
{
    tbl.EndUpdate();
}
tbl.Cell(2, 1).Split(1, 3);
```

Visual Basic

```
(TablesActions.vb)
Dim document As Document = server.Document
' Insert new table.
Dim tbl As Table = document.Tables.Create(document.Range.Start, 1, 3, Auto
' Create a table header.
document.InsertText(tbl(0, 0).Range.Start, "Name")
document.InsertText(tbl(0, 1).Range.Start, "Size")
document.InsertText(tbl(0, 2).Range.Start, "DateTime")
' Insert table data.
Dim dirinfo As New DirectoryInfo("C:\")
Try
    tbl.BeginUpdate()
    For Each fi As FileInfo In dirinfo.GetFiles()
        Dim row As TableRow = tbl.Rows.Append()
        Dim cell As TableCell = row.FirstCell
        Dim fileName As String = fi.Name
        Dim fileLength As String = String.Format("{0:N0}", fi.Length)
        Dim fileLastTime As String = String.Format("{0:g}", fi.LastWriteTi
        document.InsertSingleLineText(cell.Range.Start, fileName)
        document.InsertSingleLineText(cell.Next.Range.Start, fileLength)
        document.InsertSingleLineText(cell.Next.Next.Range.Start, fileLast
    Next fi
    ' Center the table header.
    For Each p As Paragraph In document.Paragraphs.Get(tbl.FirstRow.Range)
        p.Alignment = ParagraphAlignment.Center
    Next p
Finally
    tbl.EndUpdate()
End Try
tbl.Cell(2, 1).Split(1, 3)
```

How to: Create a Table with Fixed Column Width

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Tables](#) > [How to: Create a Table with Fixed Column Width](#)

To create a table with fixed column widths, specify the following settings:

Property	Value
For the table set the Table.TableLayout	TableLayoutType.Fixed
For the table set the Table.PreferredWidthType	WidthType.Fixed
For each cell set the TableCell.PreferredWidthType	WidthType.Fixed
For each cell set the TableCell.PreferredWidth	A floating point value specifying the cell width measured in document units of measurement determined by the Document.Unit property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(TablesActions.cs)
Document document = server.Document;
Table table = document.Tables.Create(document.Range.Start, 3, 3);
table.TableAlignment = TableRowAlignment.Center;
table.TableLayout = TableLayoutType.Fixed;
table.PreferredWidthType = WidthType.Fixed;
table.PreferredWidth = DevExpress.Office.Utils.Units.InchesToDocumentsF(4f);
table.Rows[1].HeightType = HeightType.Exact;
table.Rows[1].Height = DevExpress.Office.Utils.Units.InchesToDocumentsF(0.8f);
table[1, 1].PreferredWidthType = WidthType.Fixed;
table[1, 1].PreferredWidth = DevExpress.Office.Utils.Units.InchesToDocumentsF(1.5f);
table.MergeCells(table[1, 1], table[2, 1]);
```

Visual Basic

```
(TablesActions.vb)
Dim document As Document = server.Document
Dim table As Table = document.Tables.Create(document.Range.Start, 3, 3)
table.TableAlignment = TableRowAlignment.Center
table.TableLayout = TableLayoutType.Fixed
table.PreferredWidthType = WidthType.Fixed
table.PreferredWidth = DevExpress.Office.Utils.Units.InchesToDocumentsF(4F)
table.Rows(1).HeightType = HeightType.Exact
table.Rows(1).Height = DevExpress.Office.Utils.Units.InchesToDocumentsF(0.
table(1, 1).PreferredWidthType = WidthType.Fixed
table(1, 1).PreferredWidth = DevExpress.Office.Utils.Units.InchesToDocumen
table.MergeCells(table(1, 1), table(2, 1))
```

How to: Merge and Split Table Cells

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Tables](#) > [How to: Merge and Split Table Cells](#)

The following example illustrates how to merge and split table cells programmatically.

Merge Cells

To merge cells, use the Table.MergeCells method with the passed cells marking a range to be merged. As a result, all the cells falling to the stated range will be combined into a single cell. The content of all the merged cells will persist.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

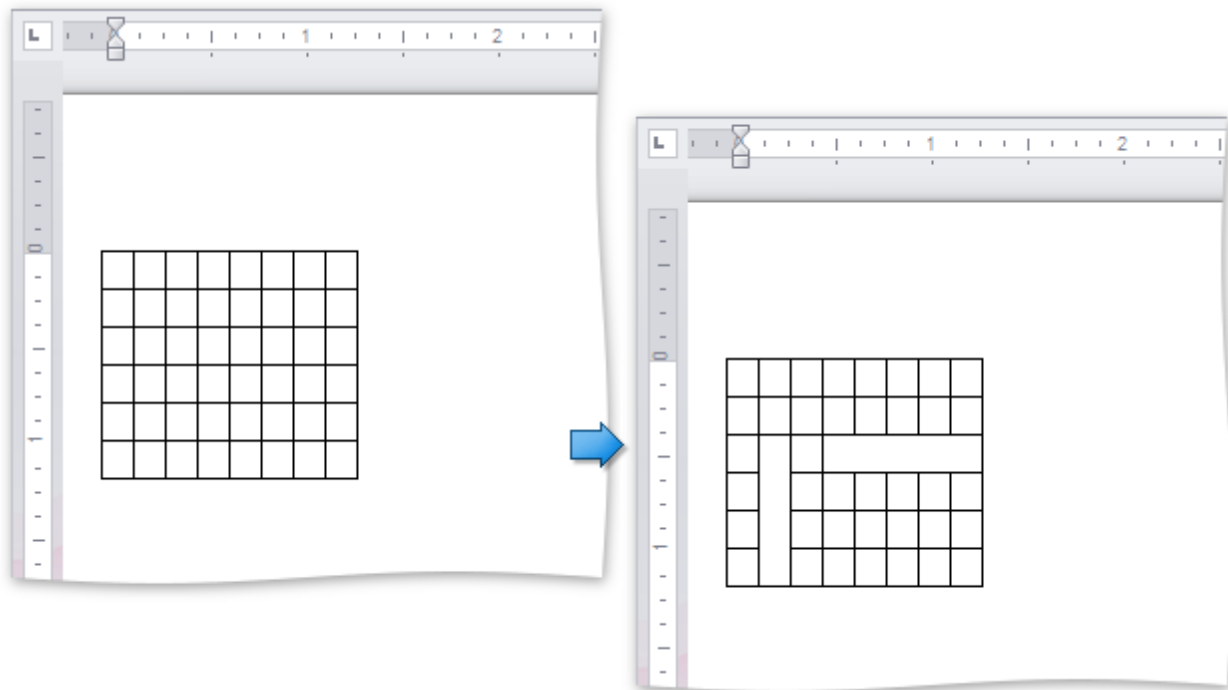
C#

(TablesActions.cs)
Document document = server.Document;
Table table = document.Tables.Create(document.Range.Start, 6, 8);
table.BeginUpdate();
table.MergeCells(table[2, 1], table[5, 1]);
table.MergeCells(table[2, 3], table[2, 7]);
table.EndUpdate();

Visual Basic

(TablesActions.vb)
Dim document As Document = server.Document
Dim table As Table = document.Tables.Create(document.Range.Start, 6, 8)
table.BeginUpdate()
table.MergeCells(table(2, 1), table(5, 1))
table.MergeCells(table(2, 3), table(2, 7))
table.EndUpdate()

The image below illustrates the result of code execution.



Split Cells

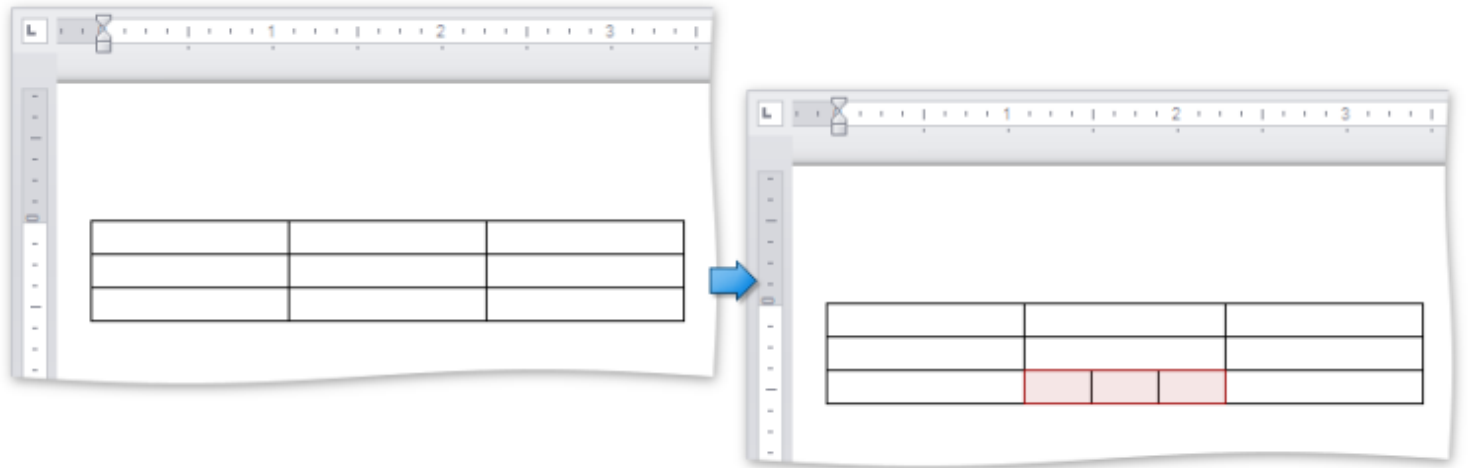
Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

The following code snippet splits a table cell into three by using the TableCell.Split method.

C#	
<pre>(TablesActions.cs) Document document = server.Document; Table table = document.Tables.Create(document.Range.Start, 3, 3, AutoFitBe //split a cell to three: table.Cell(2, 1).Split(1, 3);</pre>	
Visual Basic	
<pre>(TablesActions.vb) Dim document As Document = server.Document Dim table As Table = document.Tables.Create(document.Range.Start, 3, 3, Au 'split a cell to three: table.Cell(2, 1).Split(1, 3)</pre>	

The result is illustrated below.



How to: Change Table Color

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Tables](#) > [How to: Change Table Color](#)

The following example illustrates how to set the color of different table elements.

To achieve the required result, the following properties can be used:

- `TableCell.BackgroundColor` - sets the cell background color;
- `TableCellBorder.LineColor` - sets the cell border color;
- `Table.TableBackgroundColor` - sets the color of the empty space between cells.

The first two properties apply color settings to a single cell. To set them to the whole table, use the `Table.ForEachCell` method. It requires a delegate to execute in each cell of a table. The delegate is represented by the `TableCellProcessorDelegate` object.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(TablesActions.cs)
Document document = server.Document;
// Create a table.
Table table = document.Tables.Create(document.Range.Start, 3, 5, AutoFitBehaviorType.AutoFitToWindow);
table.BeginUpdate();
// Provide the space between table cells.
// The distance between cells will be 4 mm.
document.Unit = DevExpress.Office.DocumentUnit.Millimeter;
table.TableCellSpacing = 2;
// Change the color of empty space between cells.
table.TableBackgroundColor = Color.Violet;
//Change cell background color.
table.ForEachCell(new TableCellProcessorDelegate(TableHelper.ChangeCellColor));
table.ForEachCell(new TableCellProcessorDelegate(TableHelper.ChangeCellBorderColor));
table.EndUpdate();

Visual Basic

(TablesActions.vb)
Dim document As Document = server.Document
' Create a table.
Dim table As Table = document.Tables.Create(document.Range.Start, 3, 5, Au
table.BeginUpdate()
' Provide the space between table cells.
' The distance between cells will be 4 mm.
document.Unit = DevExpress.Office.DocumentUnit.Millimeter
table.TableCellSpacing = 2
' Change the color of empty space between cells.
table.TableBackgroundColor = Color.Violet
'Change cell background color.
table.ForEachCell(New TableCellProcessorDelegate(AddressOf TableHelper.Cha
table.ForEachCell(New TableCellProcessorDelegate(AddressOf TableHelper.Cha
table.EndUpdate())

As a result, the table will look as illustrated below.



How to: Create and Apply Table Style

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Tables](#) > [How to: Create and Apply Table Style](#)

This example describes how to create new table style and apply it to the table.

To create a new table style, do the following:

1. Create a new TableStyle object using the TableStyleCollection.CreateNew method.
2. Specify the style settings using the corresponding TableStyle properties.
3. Add a new item in the table styles collection, represented by the TableStyleCollection instance. To do that, access the collection by the Document.TableStyles property and use the TableCellStyleCollection.Add method. Note that without adding a table style to the collection, you won't be able to apply it.
4. To apply the newly created style to the table, set the table's Table.Style property to the corresponding object.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(TablesActions.cs)
Document document = server.Document;
document.BeginUpdate();
// Create a new table style.
TableStyle tStyleMain = document.TableStyles.CreateNew();
// Specify style characteristics.
tStyleMain.AllCaps = true;
tStyleMain.FontName = "Segoe Condensed";
tStyleMain.FontSize = 14;
tStyleMain.Alignment = ParagraphAlignment.Center;
tStyleMain.TableBorders.InsideHorizontalBorder.LineStyle = TableBorderLineStyle.Dotted;
tStyleMain.TableBorders.InsideVerticalBorder.LineStyle = TableBorderLineStyle.Dotted;
tStyleMain.TableBorders.Top.LineThickness = 1.5f;
tStyleMain.TableBorders.Top.LineStyle = TableBorderLineStyle.Double;
tStyleMain.TableBorders.Left.LineThickness = 1.5f;
tStyleMain.TableBorders.Left.LineStyle = TableBorderLineStyle.Double;
tStyleMain.TableBorders.Bottom.LineThickness = 1.5f;
tStyleMain.TableBorders.Bottom.LineStyle = TableBorderLineStyle.Double;
tStyleMain.TableBorders.Right.LineThickness = 1.5f;
tStyleMain.TableBorders.Right.LineStyle = TableBorderLineStyle.Double;
tStyleMain.CellBackgroundColor = System.Drawing.Color.LightBlue;
tStyleMain.TableLayout = TableLayoutType.Fixed;
tStyleMain.Name = "MyTableStyle";
//Add the style to the document.
document.TableStyles.Add(tStyleMain);
document.EndUpdate();
document.BeginUpdate();
// Create a table.
Table table = document.Tables.Create(document.Range.Start, 3, 3);
table.TableLayout = TableLayoutType.Fixed;
table.PreferredWidthType = WidthType.Fixed;
table.PreferredWidth = DevExpress.Office.Utils.Units.InchesToDocumentsF(4.5f);
table[1, 1].PreferredWidthType = WidthType.Fixed;
table[1, 1].PreferredWidth = DevExpress.Office.Utils.Units.InchesToDocumentsF(1.5f);
// Apply a previously defined style.
table.Style = tStyleMain;
document.EndUpdate();
document.InsertText(table[1, 1].Range.Start, "STYLED");
```

Visual Basic

```
(TablesActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
' Create a new table style.
Dim tStyleMain As TableStyle = document.TableStyles.CreateNew()
' Specify style characteristics.
tStyleMain.AllCaps = True
tStyleMain.FontName = "Segoe Condensed"
tStyleMain.FontSize = 14
tStyleMain.Alignment = ParagraphAlignment.Center
tStyleMain.TableBorders.InsideHorizontalBorder.LineStyle = TableBorderLineStyle.Double
tStyleMain.TableBorders.InsideVerticalBorder.LineStyle = TableBorderLineStyle.Double
tStyleMain.TableBorders.Top.LineThickness = 1.5F
tStyleMain.TableBorders.Top.LineStyle = TableBorderLineStyle.Double
tStyleMain.TableBorders.Left.LineThickness = 1.5F
tStyleMain.TableBorders.Left.LineStyle = TableBorderLineStyle.Double
tStyleMain.TableBorders.Bottom.LineThickness = 1.5F
tStyleMain.TableBorders.Bottom.LineStyle = TableBorderLineStyle.Double
tStyleMain.TableBorders.Right.LineThickness = 1.5F
tStyleMain.TableBorders.Right.LineStyle = TableBorderLineStyle.Double
tStyleMain.CellBackgroundColor = System.Drawing.Color.LightBlue
tStyleMain.TableLayout = TableLayoutType.Fixed
tStyleMain.Name = "MyTableStyle"
'Add the style to the document.
document.TableStyles.Add(tStyleMain)
document.EndUpdate()
document.BeginUpdate()
' Create a table.
Dim table As Table = document.Tables.Create(document.Range.Start, 3, 3)
table.TableLayout = TableLayoutType.Fixed
table.PreferredWidthType = WidthType.Fixed
table.PreferredWidth = DevExpress.Office.Utils.Units.InchesToDocumentsF(4.
table(1, 1).PreferredWidthType = WidthType.Fixed
table(1, 1).PreferredWidth = DevExpress.Office.Utils.Units.InchesToDocumentsF(4.
' Apply a previously defined style.
table.Style = tStyleMain
document.EndUpdate()
document.InsertText(table(1, 1).Range.Start, "STYLED")
```

The code execution result is illustrated below.



How to: Use Conditional Style

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Tables](#) > [How to: Use Conditional Style](#)

The Rich Edit API allows you to set formatting rules to the specific table elements (first row, first cell, odd column, etc.). This feature is called *conditional styles*.

To create and apply a conditional style, perform the following steps.

1.

Create new table style using the `TableStyleCollection.CreateNew` method.
2.

Optionally, specify the newly created style's `TableStyle.Parent` property to derive it from an existing style.
3.

Create separate `TableConditionalStyle` objects for each desired table element type that needs to be formatted. To do so, call the `TableConditionalStyleProperties.CreateConditionalStyle` method repeatedly and pass different `ConditionalTableStyleFormattingTypes` enumeration values to this method as parameters.
4.

Specify additional settings for each `TableConditionalStyle` object you have:
 - `TableCellPropertiesBase.CellBackgroundColor` - sets the cell background color;
 - `TableCellBorder.LineColor` - sets the cell border color;
 - `Table.TableBackgroundColor` - sets the color of the empty space between cells.
 - `CharacterPropertiesBase.FontName` - sets the content's font;
 - `CharacterPropertiesBase.FontSize` - sets the content's font size;
 - `CharacterPropertiesBase.ForeColor` - sets the content's font color;
 - `ParagraphPropertiesBase.Alignment` - sets the content's alignment, etc.
5.

Add the style created in step 1 to the table styles collection by calling the `TableCellStyleCollection.Add` method. Note that styles not added to a corresponding collection cannot be applied to Rich Edit elements.
6.

To apply the style to a table, set this table's `Table.Style` property.
7.

Specify the table elements that should be formatted using the `Table.TableLook` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

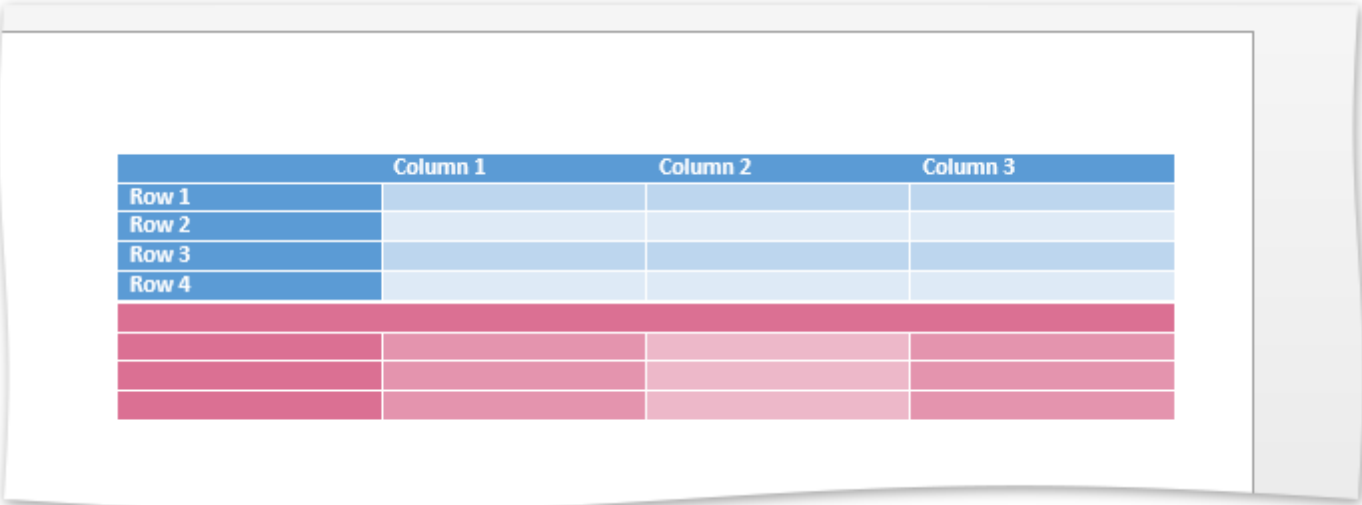
C#

```
(TablesActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\TableStyles.docx", DocumentFormat.OpenXml);
document.BeginUpdate();
// Create a new style that is based on the 'Grid Table 5 Dark Accent 1' style defined in the loaded document
TableStyle myNewStyle = document.TableStyles.CreateNew();
myNewStyle.Parent = document.TableStyles["Grid Table 5 Dark Accent 1"];
// Create conditional styles (styles for table elements)
TableConditionalStyle myNewStyleForFirstRow =
    myNewStyle.ConditionalStyleProperties.CreateConditionalStyle(ConditionalTableStyleFormattingTypes.FirstRow);
myNewStyleForFirstRow.CellBackgroundColor = Color.PaleVioletRed;
TableConditionalStyle myNewStyleForFirstColumn =
    myNewStyle.ConditionalStyleProperties.CreateConditionalStyle(ConditionalTableStyleFormattingTypes.FirstColumn);
myNewStyleForFirstColumn.CellBackgroundColor = Color.PaleVioletRed;
TableConditionalStyle myNewStyleForOddColumns =
    myNewStyle.ConditionalStyleProperties.CreateConditionalStyle(ConditionalTableStyleFormattingTypes.OddColumns);
myNewStyleForOddColumns.CellBackgroundColor = System.Windows.Forms.ControlPaint.Light(Color.PaleVioletRed);
TableConditionalStyle myNewStyleForEvenColumns =
    myNewStyle.ConditionalStyleProperties.CreateConditionalStyle(ConditionalTableStyleFormattingTypes.EvenColumns);
myNewStyleForEvenColumns.CellBackgroundColor = System.Windows.Forms.ControlPaint.LightLight(Color.PaleVioletRed);
document.TableStyles.Add(myNewStyle);
// Create a new table and apply a new style.
Table table = document.Tables.Create(document.Range.End, 4, 4, AutoFitBehaviorType.AutoFitToWindow);
table.Style = myNewStyle;
// Specify which conditional styles are in effect.
table.TableLook = TableLookTypes.ApplyFirstRow | TableLookTypes.ApplyFirstColumn;
document.EndUpdate();
```

Visual Basic

```
(TablesActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\TableStyles.docx", DocumentFormat.OpenXml)
document.BeginUpdate()
' Create a new style that is based on the 'Grid Table 5 Dark Accent 1' style
Dim myNewStyle As TableStyle = document.TableStyles.CreateNew()
myNewStyle.Parent = document.TableStyles("Grid Table 5 Dark Accent 1")
' Create conditional styles (styles for table elements)
Dim myNewStyleForFirstRow As TableConditionalStyle = myNewStyle.ConditionalStyles.AddNew()
myNewStyleForFirstRow.CellBackgroundColor = Color.PaleVioletRed
Dim myNewStyleForFirstColumn As TableConditionalStyle = myNewStyle.ConditionalStyles.AddNew()
myNewStyleForFirstColumn.CellBackgroundColor = Color.PaleVioletRed
Dim myNewStyleForOddColumns As TableConditionalStyle = myNewStyle.ConditionalStyles.AddNew()
myNewStyleForOddColumns.CellBackgroundColor = System.Windows.Forms.ControlColors.ControlText
Dim myNewStyleForEvenColumns As TableConditionalStyle = myNewStyle.ConditionalStyles.AddNew()
myNewStyleForEvenColumns.CellBackgroundColor = System.Windows.Forms.ControlColors.ControlText
document.TableStyles.Add(myNewStyle)
' Create a new table and apply a new style.
Dim table As Table = document.Tables.Create(document.Range.End, 4, 4, AutoFitWidth)
table.Style = myNewStyle
' Specify which conditional styles are in effect.
table.TableLook = TableLookTypes.ApplyFirstRow Or TableLookTypes.ApplyFirstColumn
document.EndUpdate()
```

The image below illustrates the result of code execution.



How to: Delete a Table

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Tables](#) > [How to: Delete a Table](#)

The following example illustrates how to delete a table cell, row or an entire table.

- To delete a single cell, use the TableCell.Delete method. The required cell will be deleted and the next one will be moved to the left.
- The TableRow.Delete method deletes a specified row from the table.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(TablesActions.cs)
Document document = server.Document;
Table tbl = document.Tables.Create(document.Range.Start, 3, 3, AutoFitBehaviorType.AutoFitToWindow);
tbl.BeginUpdate();
tbl.Rows[2].Delete();
tbl.Cell(1, 1).Delete();
tbl.EndUpdate();
```

Visual Basic

```
(TablesActions.vb)
Dim document As Document = server.Document
Dim tbl As Table = document.Tables.Create(document.Range.Start, 3, 3, Auto
tbl.BeginUpdate()
tbl.Rows(2).Delete()
tbl.Cell(1, 1).Delete()
tbl.EndUpdate()
```

- To delete the whole table, delete it from the collection of tables contained in the current document. To do that, use the TableCollection.Remove method.

C#

```
Table tbl = document.Tables.Create(document.Range.Start, 3, 4);
//To delete the table, uncomment the method below:
// document.Tables.Remove(tbl);
```

Visual Basic

```
Dim tbl As Table = document.Tables.Create(document.Range.Start, 3, 4)
'To delete the table, uncomment the method below:
' document.Tables.Remove(tbl);
```

Document Elements

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Elements](#)

This section contains the following examples:

- [How to: Insert Bookmark or Hyperlink](#)
- [How to: Create, Edit and Delete Comments](#)
- [How to: Create and Modify Header](#)
- [How to: Specify Document Properties](#)
- [How to: Insert and Modify Fields](#)
- [How to: Create a Checkbox](#)

How to: Insert Bookmark or Hyperlink

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Elements](#) > [How to: Insert Bookmark or Hyperlink](#)

The following example demonstrates how to create a bookmark or hyperlink in the document.

Create Hyperlink

To create a hyperlink, use the `HyperlinkCollection.Create` method. The newly created `Hyperlink` object will be automatically added to the document hyperlink collection, represented by the `HyperlinkCollection` instance.

Use the hyperlink's `Hyperlink.NavigateUri` property to set a target URI. To specify a text for the tooltip displayed when the mouse hovers over a hyperlink, use the `Hyperlink.ToolTip` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(BookmarksAndHyperlinks.cs)
Document document = server.Document;
DocumentPosition hPos = server.Document.Range.Start;
server.Document.Hyperlinks.Create(document.InsertText(hPos, "Follow me!"));
document.Hyperlinks[0].NavigateUri = "https://www.devexpress.com/Products/NET/Controls/WinForms/Rich_Edit
document.Hyperlinks[0].ToolTip = "WinForms Rich Text Editor";
```

Visual Basic

```
(BookmarksAndHyperlinks.vb)
Dim document As Document = server.Document
Dim hPos As DocumentPosition = server.Document.Range.Start
server.Document.Hyperlinks.Create(document.InsertText(hPos, "Follow me!"))
document.Hyperlinks(0).NavigateUri = "https://www.devexpress.com/Products/NET/Controls/WinForms/Rich_Edit
document.Hyperlinks(0).ToolTip = "WinForms Rich Text Editor"
```

Create Bookmark

To create a bookmark, use the `BookmarkCollection.Create` method. It creates a `Bookmark` object with the specified name and range (or position) and adds it to the `BookmarkCollection`. Use the `Document.Bookmark` property to get access to the collection.

A bookmark can be referenced by a hyperlink. To do that, set the hyperlink's `Hyperlink.Anchor` property to the corresponding bookmark name, as in the code snippet below.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#	
----	--

```
(BookmarksAndHyperlinks.cs)
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
server.BeginUpdate();
Document document = server.Document;
DocumentPosition pos = document.Range.Start;
document.Bookmarks.Create(server.Document.CreateRange(pos, 0), "Top");
//Insert the hyperlink anchored to the created bookmark:
DocumentPosition pos1 = document.CreatePosition((server.Document.Range.End
document.Hyperlinks.Create(server.Document.InsertText(pos1, "get to the to
document.Hyperlinks[0].Anchor = "Top";
server.EndUpdate();
```

Visual Basic

```
(BookmarksAndHyperlinks.vb)
server.LoadDocument("Documents\Grimm.docx", DocumentFormat.OpenXml)
server.BeginUpdate()
Dim document As Document = server.Document
Dim pos As DocumentPosition = document.Range.Start
document.Bookmarks.Create(server.Document.CreateRange(pos, 0), "Top")
'Insert the hyperlink anchored to the created bookmark:
Dim pos1 As DocumentPosition = document.CreatePosition((server.Document.Ra
document.Hyperlinks.Create(server.Document.InsertText(pos1, "get to the to
document.Hyperlinks(0).Anchor = "Top"
server.EndUpdate()
```

How to: Create, Edit and Delete Comments

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Elements](#) > [How to: Create, Edit and Delete Comments](#)

The following example illustrates how to add, edit and delete comments in the current document.

All the document comments are contained in the collection represented by the `CommentCollection` interface. The collection can be accessed through the `SubDocument.Comments` property.

Add a comment

- To create a comment, use `CommentCollection.Create` method. It allows you to create a comment with the specified settings, such as author or creation date.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.



C#

(CommentsActions.cs)
Document document = server.Document;
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
DocumentRange docRange = document.Paragraphs[2].Range;
string commentAuthor = "Johnson Alphonso D";
document.Comments.Create(docRange, commentAuthor, DateTime.Now);

Visual Basic

(CommentsActions.vb)
Dim document As Document = server.Document
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim docRange As DocumentRange = document.Paragraphs(2).Range
Dim commentAuthor As String = "Johnson Alphonso D"
document.Comments.Create(docRange, commentAuthor, Date.Now)

To create a nested comment, use the `CommentCollection.Create` method with the parent comment set as an argument.

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=T418535>.**C#**

```
(CommentsActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
if (document.Comments.Count > 0)
{
    DocumentRange[] resRanges = document.FindAll("trump", SearchOptions.No
    if (resRanges.Length > 0)
    {
        Comment newComment = document.Comments.Create("Vicars Anny", docum
        newComment.Date = DateTime.Now;
    }
}
```

Visual Basic

```
(CommentsActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
If document.Comments.Count > 0 Then
    Dim resRanges() As DocumentRange = document.FindAll("trump", SearchOpt
    If resRanges.Length > 0 Then
        Dim newComment As Comment = document.Comments.Create("Vicars Anny"
        newComment.Date = Date.Now
    End If
End If
```

Edit a comment

To edit the comment content, use the `Comment.BeginUpdate` and `Comment.EndUpdate` paired methods. You can add any text and insert additional objects to the comment, as illustrated below.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```

(CommentsActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
int commentCount = document.Comments.Count;
if (commentCount > 0)
{
    Comment comment = document.Comments[document.Comments.Count - 1];
    if (comment != null)
    {
        SubDocument commentDocument = comment.BeginUpdate();
        commentDocument.InsertText(commentDocument.CreatePosition(0), "som
        commentDocument.Tables.Create(commentDocument.CreatePosition(9), 5
        comment.EndUpdate(commentDocument);
    }
}

```

Visual Basic

```

(CommentsActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim commentCount As Integer = document.Comments.Count
If commentCount > 0 Then
    Dim comment As Comment = document.Comments(document.Comments.Count - 1)
    If comment IsNot Nothing Then
        Dim commentDocument As SubDocument = comment.BeginUpdate()
        commentDocument.InsertText(commentDocument.CreatePosition(0), "som
        commentDocument.Tables.Create(commentDocument.CreatePosition(9), 5
        comment.EndUpdate(commentDocument)
    End If
End If

```

You can also change the comment settings by changing the corresponding Comment object properties. All the operation should be enclosed with Comment.BeginUpdate and Comment.EndUpdate paired methods as well.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(CommentsActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
int commentCount = document.Comments.Count;
if (commentCount > 0)
{
    document.BeginUpdate();
    Comment comment = document.Comments[document.Comments.Count - 1];
    comment.Name = "New Name";
    comment.Date = DateTime.Now;
    comment.Author = "New Author";
    document.EndUpdate();
}
```

Visual Basic

```
(CommentsActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim commentCount As Integer = document.Comments.Count
If commentCount > 0 Then
    document.BeginUpdate()
    Dim comment As Comment = document.Comments(document.Comments.Count - 1)
    comment.Name = "New Name"
    comment.Date = Date.Now
    comment.Author = "New Author"
    document.EndUpdate()
End If
```

Delete a comment

To delete a comment from the document, delete it from the corresponding collection. To do that, use the `CommentCollection.Remove` method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(CommentsActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
if (document.Comments.Count > 0)
{
    document.Comments.Remove(document.Comments[0]);
}
```

Visual Basic

```
(CommentsActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\Grimm.docx", DocumentFormat.OpenXml)
If document.Comments.Count > 0 Then
    document.Comments.Remove(document.Comments(0))
End If
```

How to: Create and Modify Header

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Elements](#) > [How to: Create and Modify Header](#)

The following article explains how to create and modify a document header.

Create a Header

To create a document header, select the section where it is going to be located. To check whether this section already has a header, use the `Section.HasHeader` property.

The `Section.BeginUpdateHeader` method is used to start the process. It also creates a new header of the specified `HeaderFooterType` type. If the type is not specified, the `HeaderFooterType.Primary` header will be created.

To work with the document header, use the `Document.ChangeActiveDocument` method. Note that if this method is not used, the wrong header can be selected and the follow-up actions may fail.

Use the `Section.EndUpdateHeader` to finish working with the header.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(HeaderAndFooterActions.cs)
Document document = server.Document;
Section firstSection = document.Sections[0];
// Create an empty header.
SubDocument newHeader = firstSection.BeginUpdateHeader();
firstSection.EndUpdateHeader(newHeader);
// Check whether the document already has a header (the same header for all pages).
if (firstSection.HasHeader(HeaderFooterType.Primary))
{
    SubDocument headerDocument = firstSection.BeginUpdateHeader();
    document.ChangeActiveDocument(headerDocument);
    document.CaretPosition = headerDocument.CreatePosition(0);
    firstSection.EndUpdateHeader(headerDocument);
}
```

Visual Basic

```
(HeaderAndFooterActions.vb)
Dim document As Document = server.Document
Dim firstSection As Section = document.Sections(0)
' Create an empty header.
Dim newHeader As SubDocument = firstSection.BeginUpdateHeader()
firstSection.EndUpdateHeader(newHeader)
' Check whether the document already has a header (the same header for all
If firstSection.HasHeader(HeaderFooterType.Primary) Then
    Dim headerDocument As SubDocument = firstSection.BeginUpdateHeader()
    document.ChangeActiveDocument(headerDocument)
    document.CaretPosition = headerDocument.CreatePosition(0)
    firstSection.EndUpdateHeader(headerDocument)
End If
```

Modify Header

To modify the header, specify the section to which the header belongs and use the `Section.BeginUpdateHeader` method to open it for modifying. To insert additional content

to the header, use the `SubDocument.InsertText` or `ShapeCollection.InsertPicture` method. Use the `Section.EndUpdateHeader` to complete the modification.

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=T418535>.

C#

```
(HeaderAndFooterActions.cs)
Document document = server.Document;
document.AppendSection();
Section firstSection = document.Sections[0];
// Modify the header of the HeaderFooterType.First type.
SubDocument myHeader = firstSection.BeginUpdateHeader(HeaderFooterType.Fir
DocumentRange range = myHeader.InsertText(myHeader.CreatePosition(0), " PA
Field fld = myHeader.Fields.Create(range.End, "PAGE \\* ARABICDASH");
myHeader.Fields.Update();
firstSection.EndUpdateHeader(myHeader);
// Display the header of the HeaderFooterType.First type on the first page
firstSection.DifferentFirstPage = true;
```

Visual Basic

```
(HeaderAndFooterActions.vb)
Dim document As Document = server.Document
document.AppendSection()
Dim firstSection As Section = document.Sections(0)
' Modify the header of the HeaderFooterType.First type.
Dim myHeader As SubDocument = firstSection.BeginUpdateHeader(HeaderFooterT
Dim range As DocumentRange = myHeader.InsertText(myHeader.CreatePosition(0
Dim fld As Field = myHeader.Fields.Create(range.End, "PAGE \\* ARABICDASH")
myHeader.Fields.Update()
firstSection.EndUpdateHeader(myHeader)
' Display the header of the HeaderFooterType.First type on the first page.
firstSection.DifferentFirstPage = True
```

How to: Specify Document Properties

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Elements](#) > [How to: Specify Document Properties](#)

This example demonstrates how to append document properties in code.

Specify Standard Document Properties

Use the Document.DocumentProperties property to specify standard document properties, as illustrated below.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(DocumentPropertiesActions.cs)
server.CreateNewDocument();
Document document = server.Document;
document.BeginUpdate();
document.DocumentProperties.Creator = "John Doe";
document.DocumentProperties.Title = "Inserting Custom Properties";
document.DocumentProperties.Category = "TestDoc";
document.DocumentProperties.Description = "This code demonstrates API to modify and display standard document properties";
document.Fields.Create(document.AppendText("\nAUTHOR: ").End, "AUTHOR");
document.Fields.Create(document.AppendText("\nTITLE: ").End, "TITLE");
document.Fields.Create(document.AppendText("\nCOMMENTS: ").End, "COMMENTS");
document.Fields.Create(document.AppendText("\nCREATEDATE: ").End, "CREATEDATE");
document.Fields.Create(document.AppendText("\nCategory: ").End, "DOCPROPERTY Category");
document.Fields.Update();
document.EndUpdate();
```

Visual Basic

```
(DocumentPropertiesActions.vb)
server.CreateNewDocument()
Dim document As Document = server.Document
document.BeginUpdate()
document.DocumentProperties.Creator = "John Doe"
document.DocumentProperties.Title = "Inserting Custom Properties"
document.DocumentProperties.Category = "TestDoc"
document.DocumentProperties.Description = "This code demonstrates API to modify and display standard document properties"
document.Fields.Create(document.AppendText(vbLf & "AUTHOR: ").End, "AUTHOR")
document.Fields.Create(document.AppendText(vbLf & "TITLE: ").End, "TITLE")
document.Fields.Create(document.AppendText(vbLf & "COMMENTS: ").End, "COMMENTS")
document.Fields.Create(document.AppendText(vbLf & "CREATEDATE: ").End, "CREATEDATE")
document.Fields.Create(document.AppendText(vbLf & "Category: ").End, "DOCPROPERTY Category")
document.Fields.Update()
document.EndUpdate()
```

Create Custom Document Properties

To create a custom document property, a new item should be added to the corresponding collection, accessible through the Document.CustomProperties property. This can be done in two ways:

- Using the DocumentCustomProperties.Item property. It sets a value to the property with the name specified in square brackets. If the property with such name doesn't exist, it will be created automatically.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(DocumentPropertiesActions.cs)
server.CreateNewDocument();
Document document = server.Document;
document.BeginUpdate();
document.Fields.Create(document.AppendText("\nMyNumericProperty: ").End, "
document.Fields.Create(document.AppendText("\nMyStringProperty: ").End, "D
document.Fields.Create(document.AppendText("\nMyBooleanProperty: ").End, "
document.EndUpdate();
document.CustomProperties.Add("MyNumericProperty", 123.45);
document.CustomProperties.Add("MyStringProperty", "The Final Answer");
document.CustomProperties.Add("MyBooleanProperty", true);
server.CalculateDocumentVariable += DocumentPropertyDisplayHelper.OnCalcul
document.Fields.Update();
```

Visual Basic

```
(DocumentPropertiesActions.vb)
server.CreateNewDocument()
Dim document As Document = server.Document
document.BeginUpdate()
document.Fields.Create(document.AppendText(vbLf & "MyNumericProperty: ").E
document.Fields.Create(document.AppendText(vbLf & "MyStringProperty: ").En
document.Fields.Create(document.AppendText(vbLf & "MyBooleanProperty: ").E
document.EndUpdate()
document.CustomProperties.Add("MyNumericProperty", 123.45)
document.CustomProperties.Add("MyStringProperty", "The Final Answer")
document.CustomProperties.Add("MyBooleanProperty", True)
AddHandler server.CalculateDocumentVariable, AddressOf DocumentPropertyDis
document.Fields.Update()
```

Using DocumentCustomProperties.Add method, as illustrated below.

C#

```
private void SetCustomProperties()
{
    Document document = server.Document;
    document.CustomProperties.Add("MyNumericProperty", 123.45);
    document.CustomProperties.Add("MyStringProperty", "The Final Answer"
    document.CustomProperties.Add("MyBookmarkProperty", document.Bookmar
    document.CustomProperties.Add("MyBooleanProperty", true);
}
```

Visual Basic

```
Private Sub SetCustomProperties()  
    Dim document As Document = server.Document  
    document.CustomProperties.Add("MyNumericProperty", 123.45)  
    document.CustomProperties.Add("MyStringProperty", "The Final Answer")  
    document.CustomProperties.Add("MyBookmarkProperty", document.Bookmarks  
    document.CustomProperties.Add("MyBooleanProperty", True)  
End Sub
```

How to: Insert and Modify Fields

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Elements](#) > [How to: Insert and Modify Fields](#)

The following example illustrates how to add and modify fields manually.

All the fields in the document are contained in the FieldCollection, accessible through the SubDocument.Fields property.

Insert a Field

Use the FieldCollection.Create method to create a field, as illustrated below.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(FieldActions.cs)
Document document = server.Document;
document.BeginUpdate();
document.Fields.Create(document.CaretPosition, "DATE");
document.Fields.Update();
document.EndUpdate();

Visual Basic

(FieldActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
document.Fields.Create(document.CaretPosition, "DATE")
document.Fields.Update()
document.EndUpdate()

Pass the range to the FieldCollection.Create method to convert it to a field.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(FieldActions.cs)
Document document = server.Document;
document.BeginUpdate();
document.AppendText("SYMBOL 0x54 \\f Wingdings \\s 24");
document.EndUpdate();
document.Fields.Create(document.Paragraphs[0].Range);
document.Fields.Update();

Visual Basic

(FieldActions.vb)
Dim document As Document = server.Document
document.BeginUpdate()
document.AppendText("SYMBOL 0x54 \f Wingdings \s 24")
document.EndUpdate()
document.Fields.Create(document.Paragraphs(0).Range)
document.Fields.Update()

Modify Field Code

Use the SubDocument.BeginUpdate - SubDocument.EndUpdate paired methods to enable modifying field content.

Call the SubDocument.InsertText method to insert a format switch. Use the end position of the field code range as a method argument.

Call the Field.Update method to update the field result.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#
<pre>(FieldActions.cs) Document document = server.Document; document.BeginUpdate(); document.Fields.Create(document.CaretPosition, "DATE"); document.EndUpdate(); for (int i = 0; i < document.Fields.Count; i++) { string fieldCode = document.GetText(document.Fields[i].CodeRange); if (fieldCode == "DATE") { DocumentPosition position = document.Fields[i].CodeRange.End; document.InsertText(position, @" "M / d / yyyy HH: mm:ss"); } } document.Fields.Update();</pre>

Visual Basic
<pre>(FieldActions.vb) Dim document As Document = server.Document document.BeginUpdate() document.Fields.Create(document.CaretPosition, "DATE") document.EndUpdate() For i As Integer = 0 To document.Fields.Count - 1 Dim fieldCode As String = document.GetText(document.Fields(i).CodeRange) If fieldCode = "DATE" Then Dim position As DocumentPosition = document.Fields(i).CodeRange.End document.InsertText(position, " \@ "M / d / yyyy HH: mm:ss") End If Next i document.Fields.Update()</pre>

Use the Field.ShowCodes property to change the target field's display mode.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

This code snippet displays field codes for all fields in the main document body. The fields containing in the header or footer belong to a different FieldCollection. Use the Section.BeginUpdateHeader- Section.EndUpdateHeader or Section.BeginUpdateFooter - Section.EndUpdateFooter paired methods to retrieve the header or footer document.

C#

```
(FieldActions.cs)
Document document = server.Document;
document.LoadDocument("MailMergeSimple.docx", DocumentFormat.OpenXml);
for (int i = 0; i < document.Fields.Count; i++)
{
    document.Fields[i].ShowCodes = true;
}
```

Visual Basic

```
(FieldActions.vb)
Dim document As Document = server.Document
document.LoadDocument("MailMergeSimple.docx", DocumentFormat.OpenXml)
For i As Integer = 0 To document.Fields.Count - 1
    document.Fields(i).ShowCodes = True
Next i
```

See Also
[Fields](#)

How to: Create a Checkbox

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Document Elements](#) > [How to: Create a Checkbox](#)

Follow the steps below to create a CheckBox object in code:

- Note

Checkboxes cannot be inserted to the document header, footer, comments or floating objects (text boxes or shapes). Otherwise, an **FormFieldIncorrectSubDocumentException** occurs.
1. Access the document's form fields collection using the SubDocument.FormFields property and call the FormFieldCollection.InsertCheckBox method to create a CheckBox object at the specified document position. In the example below, the checkbox is created at the caret's position.
 2. Define the name of the bookmark associated with the current CheckBox object using the FormField.Name property.
 3. Specify the checkbox's state using the CheckBox.State property.
 4. Use the CheckBox.SizeMode property to define the checkbox's size. Selecting the CheckBoxSizeMode.Auto mode resizes the checkbox according to the current font size value (returned by the CharacterPropertiesBase.FontSize property). Set the **SizeMode** property to CheckBoxSizeMode.Exact and specify the CheckBox.Size value to specify the checkbox's size.
 5. You can add instructional text to the checkbox. This text can be displayed in the status bar or when focusing the checkbox and pressing **F1**. Specify the help text type and the text using the following members:
 - *Help text displayed on the status bar:* FormField.StatusTextType and FormField.StatusText.
 - *Help text displayed on pressing **F1**:* FormField.HelpTextType and FormField.HelpText.

When setting the **HelpTextType** or **StatusTextType** property to FormFieldTextType.Auto, make sure that the provided **HelpText** or **StatusText** value is equal to one of the AutoText gallery's entries (stored as the **Normal.dotm** file in the system's **Templates** folder). Otherwise, the help text is not displayed.

Note

Currently, the RichEditControl cannot display any instructional text.

6. Set the FormField.CalculateOnExit property to **true** to update the checkbox's value on exit.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(FormFieldsActions.cs)
DocumentPosition currentPosition = server.Document.CaretPosition;
DevExpress.XtraRichEdit.API.Native.CheckBox checkBox = server.Document.FormFields.InsertCheckBox(currentPosition);
checkBox.Name = "check1";
checkBox.State = CheckBoxState.Checked;
checkBox.SizeMode = CheckBoxSizeMode.Auto;
checkBox.HelpTextType = FormFieldTextType.Custom;
checkBox.HelpText = "help text";
```

Visual Basic

```
(FormFieldsActions.vb)
Dim currentPosition As DocumentPosition = server.Document.CaretPosition
Dim checkBox As DevExpress.XtraRichEdit.API.Native.CheckBox = server.Document.FormFields.InsertCheckBox(currentPosition)
checkBox.Name = "check1"
checkBox.State = CheckBoxState.Checked
checkBox.SizeMode = CheckBoxSizeMode.Auto
checkBox.HelpTextType = FormFieldTextType.Custom
checkBox.HelpText = "help text"
```

Layout

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Layout](#)

This section contains the following examples:

- [How to: Configure the Page Layout Programmatically](#)
- [How to: Create a Three-Column Layout with Uniform Columns](#)
- [How To: Add Line Numbering](#)

How to: Configure the Page Layout Programmatically

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Layout](#) > [How to: Configure the Page Layout Programmatically](#)

This example illustrates how to adjust page settings in code.

The page layout settings in the API pertain to the document's section. Therefore, you need to change the `SectionPage` properties to adjust page settings.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

The following code specifies the page layout settings for the first section of the document: the A6 paper in landscape orientation with the left margin set to two inches.

C#

(PageLayoutActions.cs)
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
Document document = server.Document;
document.Unit = DevExpress.Office.DocumentUnit.Inch;
document.Sections[0].Page.PaperKind = System.Drawing.Printing.PaperKind.A6;
document.Sections[0].Page.Landscape = true;
document.Sections[0].Margins.Left = 2.0f;

Visual Basic

(PageLayoutActions.vb)
server.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim document As Document = server.Document
document.Unit = DevExpress.Office.DocumentUnit.Inch
document.Sections(0).Page.PaperKind = System.Drawing.Printing.PaperKind.A6
document.Sections(0).Page.Landscape = True
document.Sections(0).Margins.Left = 2.0F

How to: Create a Three-Column Layout with Uniform Columns

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Layout](#) > [How to: Create a Three-Column Layout with Uniform Columns](#)

These columns can be created using the methods provided by the SectionColumns interface, accessible vby the Section.Columns property of the document section. First, create the required column layout. The SectionColumns.CreateUniformColumns method allows you to create a specified number of columns on a page with the required spacing. Then, the layout is applied to the document content using the SectionColumns.SetColumns method.

The following code illustrates how the above technique can be used to create three columns with 0.2 inches of distance between them. They will have the same width, calculated automatically according to the current page layout.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(PageLayoutActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DevExpress.XtraRichEdit.DocumentFormat.OpenXml);
document.Unit = DevExpress.Office.DocumentUnit.Inch;
// Get the first section in a document
Section firstSection = document.Sections[0];
// Create columns and apply them to the document
SectionColumnCollection sectionColumnsLayout =
 firstSection.Columns.CreateUniformColumns(firstSection.Page, 0.2f, 3);
firstSection.Columns.SetColumns(sectionColumnsLayout);

Visual Basic

(PageLayoutActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\Grimm.docx", DevExpress.XtraRichEdit.Docu
document.Unit = DevExpress.Office.DocumentUnit.Inch
' Get the first section in a document
Dim firstSection As Section = document.Sections(0)
' Create columns and apply them to the document
Dim sectionColumnsLayout As SectionColumnCollection = firstSection.Columns
firstSection.Columns.SetColumns(sectionColumnsLayout)

How To: Add Line Numbering

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Layout](#) > [How To: Add Line Numbering](#)

The following example illustrates how to adjust line numbering to the specified document section.

To enable **Line Numbering**, set the RichEditView.AllowDisplayLineNumbers option for the current view to **true** and specify the SectionLineNumbering.CountBy property to a non-zero positive value.

In SimpleView and DraftView views line numbers are outside the default visible area, so you have to provide a space to display them by setting the left padding to a higher value (use the SimpleView.Padding or the DraftView.Padding property, respectively).

The line numbering font face and font color are specified by the **Line Number** character style.

To specify line numbering, the following properties are provided:

- SectionLineNumbering.Start - sets the starting line number;
- SectionLineNumbering.Distance - sets the distance between the line number and the start of the line;
- SectionLineNumbering.RestartType - sets when the line numbering should be reset to the starting line number;

All the line numbering parameters can be accessed through the Section.LineNumbering property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

(PageLayoutActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DevExpress.XtraRichEdit.DocumentFormat.OpenXml);
document.Unit = DevExpress.Office.DocumentUnit.Inch;
Section sec = document.Sections[0];
sec.LineNumbering.CountBy = 2;
sec.LineNumbering.Start = 1;
sec.LineNumbering.Distance = 0.25f;
sec.LineNumbering.RestartType = LineNumberingRestart.NewSection;

Visual Basic

(PageLayoutActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\Grimm.docx", DevExpress.XtraRichEdit.Docu
document.Unit = DevExpress.Office.DocumentUnit.Inch
Dim sec As Section = document.Sections(0)
sec.LineNumbering.CountBy = 2
sec.LineNumbering.Start = 1
sec.LineNumbering.Distance = 0.25F
sec.LineNumbering.RestartType = LineNumberingRestart.NewSection

Protection

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Protection](#)

This section contains the following examples:

- [How to: Protect a Document](#)
- [How to: Grant Editing Permissions in a Document](#)

How to: Protect a Document

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Protection](#) > [How to: Protect a Document](#)

Document protection prevents end-users from modifying a document (by adding, formatting or deleting the document parts, inserting tables, images, comments, etc.).
The following example demonstrates how to protect a document by calling the Document.Protect method :

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ProtectionActions.cs)
server.LoadDocument("Documents//Grimm.docx", DocumentFormat.OpenXml);
Document document = server.Document;
if (!document.IsDocumentProtected)
{
    //Protect the document with a password
    document.Protect("123", DocumentProtectionType.ReadOnly);
    //Insert a comment indicating that the document is protected
    document.Comments.Create(document.Paragraphs[0].Range, "Admin");
    SubDocument commentDocument = document.Comments[0].BeginUpdate();
    commentDocument.InsertText(commentDocument.CreatePosition(0),
        "Document is protected with a password.\nYou cannot modify the document until protection is removed.");
    commentDocument.EndUpdate();
}
```

Visual Basic

```
(ProtectionActions.vb)
server.LoadDocument("Documents//Grimm.docx", DocumentFormat.OpenXml)
Dim document As Document = server.Document
If Not document.IsDocumentProtected Then
    'Protect the document with a password
    document.Protect("123", DocumentProtectionType.ReadOnly)
    'Insert a comment indicating that the document is protected
    document.Comments.Create(document.Paragraphs(0).Range, "Admin")
    Dim commentDocument As SubDocument = document.Comments(0).BeginUpdate()
    commentDocument.InsertText(commentDocument.CreatePosition(0), "Document
commentDocument.EndUpdate()
End If
```

Unprotect a Document

Use the Document.Unprotect method to remove protection.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ProtectionActions.cs)
server.LoadDocument("Documents//Grimm_Protected.docx", DocumentFormat.Open
Document document = server.Document;
if (document.IsDocumentProtected == true)
{
    //Unprotect the document
    document.Unprotect();
    //Insert a comment indicating that the document can be edited
    document.Comments.Create(document.Paragraphs[0].Range, "Admin");
    SubDocument commentDocument = document.Comments[0].BeginUpdate();
    commentDocument.InsertText(commentDocument.CreatePosition(0),
    "Document is unprotected. You can modify the document according to your
    commentDocument.EndUpdate();
}
```

Visual Basic

```
(ProtectionActions.vb)
server.LoadDocument("Documents//Grimm_Protected.docx", DocumentFormat.Open
Dim document As Document = server.Document
If document.IsDocumentProtected = True Then
    'Unprotect the document
    document.Unprotect()
    'Insert a comment indicating that the document can be edited
    document.Comments.Create(document.Paragraphs(0).Range, "Admin")
    Dim commentDocument As SubDocument = document.Comments(0).BeginUpdate(
    commentDocument.InsertText(commentDocument.CreatePosition(0), "Documen
    commentDocument.EndUpdate()
End If
```

How to: Grant Editing Permissions in a Document

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Protection](#) > [How to: Grant Editing Permissions in a Document](#)

This code snippet illustrates how to unlock specific document ranges in a protected document for authenticated users by doing the following:

1. Access the document collection of ranges with permissions by calling the SubDocument.BeginUpdateRangePermissions method.
2. Call the RangePermissionCollection.CreateRangePermission method to create a RangePermission instance to the target document range.
3. Specify the user and/or the group of users that are eligible to edit the document using the RangePermission.Group and RangePermission.UserName properties.
4. Finish the update by calling the SubDocument.EndUpdateRangePermissions method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ProtectionActions.cs)
server.LoadDocument("Documents//Grimm.docx", DocumentFormat.OpenXml);
Document document = server.Document;
// Protect document range
RangePermissionCollection rangePermissions = document.BeginUpdateRangePermissions();
RangePermission rp = rangePermissions.CreateRangePermission(document.Paragraphs[3].Range);
rp.Group = "Everyone";
rangePermissions.Add(rp);
document.EndUpdateRangePermissions(rangePermissions);
// Enforce protection and set password.
document.Protect("123");
```

Visual Basic

```
(ProtectionActions.vb)
server.LoadDocument("Documents//Grimm.docx", DocumentFormat.OpenXml)
Dim document As Document = server.Document
' Protect document range
Dim rangePermissions As RangePermissionCollection = document.BeginUpdateRangePermissions()
Dim rp As RangePermission = rangePermissions.CreateRangePermission(document.Paragraphs[3].Range)
rp.Group = "Everyone"
rangePermissions.Add(rp)
document.EndUpdateRangePermissions(rangePermissions)
' Enforce protection and set password.
document.Protect("123")
```

Printing

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Printing](#)

This section contains the following examples:

- [How to: Print a Document](#)
- [How to: Print a Document using the PrintableComponentLink](#)
- [How to: Show a Print Preview Form](#)

How to: Print a Document

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Printing](#) > [How to: Print a Document](#)

The RichEditDocumentServer component allows you to print the document with the default settings or specify printing options. Call the RichEditDocumentServer.Print method to print a document to the default system printer, as shown in the code snippet below:

C#	
<pre>server.LoadDocument("Grimm.docx"); server.Print();</pre>	
Visual Basic	
<pre>server.LoadDocument("Grimm.docx") server.Print()</pre>	

Pass the **System.Drawing.Printing.PrinterSettings** instance as the method's *printerSettings* parameter to specify the printer settings (the page range, the number of copies, etc.), as shown below:

Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T590908 .	

C#	
<pre>(XtraForm1.cs) PrinterSettings printerSettings = new PrinterSettings(); //Set the document pages to print: printerSettings.FromPage = 2; printerSettings.ToPage = 3; //Specify the number of copies: printerSettings.Copies = 2; //Print the document: server.Print(printerSettings);</pre>	

Visual Basic	
<pre>(XtraForm1.vb) Dim printerSettings As New PrinterSettings() 'Set the document pages to print: printerSettings.FromPage = 2 printerSettings.ToPage = 3 'Specify the number of copies: printerSettings.Copies = 2 'Print the document: server.Print(printerSettings)</pre>	

Note	
<p>PrinterSettings properties, such as PageSettings.Margins or PageSettings.Landscape (accessed using the PrinterSettings.DefaultPageSettings property), do not affect a printed document's layout. Change a document section's settings (accessed using the Section.Page property) to modify the document page's layout properties before printing.</p>	

How to: Print a Document using the PrintableComponentLink

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Printing](#) > [How to: Print a Document using the PrintableComponentLink](#)

This example demonstrates how to use a PrintableComponentLink to print from the RichEditDocumentServer. An instance of the RichEditDocumentServer loads a document, inserts a datetime stamp in its header and prints it. Various print options, such as page size, orientation and margins are set in code at run time using the Section interface. The application searches a list of installed printers for a printer containing a certain text ("Canon") in its name and prints to that printer. If a printer with the specified name is not found, the default printer is used.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4846>.

Note

The following code sample uses the PrintingSystem class. Make sure that the **DevExpress.XtraPrinting.v18.1.dll** assembly is added to your project.

C#

```

(Form1.cs)
using System;
using System.Windows.Forms;
using System.Drawing.Printing;
using DevExpress.XtraPrinting;
using DevExpress.XtraRichEdit;
using DevExpress.XtraRichEdit.API.Native;
namespace RichEdit_PrintingSystem
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        #region #printfromserver
        private void btn_PrintFromServer_Click(object sender, EventArgs e)
        {
            RichEditDocumentServer srv = new RichEditDocumentServer();
            srv.LoadDocument("Grimm.docx", DocumentFormat.OpenXml);
            // Insert a field displaying the current date/time into the document header.
            srv.BeginUpdate();
            SubDocument _header = srv.Document.Sections[0].BeginUpdateHeader();
            _header.Fields.Create(_header.Range.Start, "DATE \@ \"dddd, MMMM dd, yyyy HH:mm:ss\" \\\MERGEI
            _header.Paragraphs[0].Alignment = ParagraphAlignment.Right;
            srv.Document.Sections[0].EndUpdateHeader(_header);
            // Specify page margins, orientation, etc.
            SetPrintOptions(srv);
            srv.EndUpdate();
            // Display field values instead of code.
            srv.Options.MailMerge.ViewMergedData = true;
            // Create a printable link to print a document.
            PrintViaLink(srv);
        }
        #endregion #printfromserver
        #region #setprintoptions
        private static void SetPrintOptions(IRichEditDocumentServer richedit)
        {
            foreach (Section _section in richedit.Document.Sections) {
                _section.Page.PaperKind = System.Drawing.Printing.PaperKind.A4;
                _section.Page.Landscape = false;
                _section.Margins.Left = 150f;
                _section.Margins.Right = 150f;
                _section.Margins.Top = 50f;
                _section.Margins.Bottom = 50f;
                _section.PageNumbering.NumberingFormat = NumberingFormat.CardinalText;
                _section.PageNumbering.FirstPageNumber = 0;
            }
        }
        #endregion #setprintoptions
        #region #printvialink
        private static void PrintViaLink(RichEditDocumentServer srv)
        {
            if (!srv.IsPrintingAvailable) return;
            PrintableComponentLink link = new PrintableComponentLink(new PrintingSystem());
            link.Component = srv;
            // Disable warnings.
            link.PrintingSystem.ShowMarginsWarning = false;
            link.PrintingSystem.ShowPrintStatusDialog = false;
            // Find a printer containing 'Canon' in its name.
            string printerName = String.Empty;
            for (int i = 0; i < PrinterSettings.InstalledPrinters.Count; i++) {
                string pName = PrinterSettings.InstalledPrinters[i];
                if (pName.Contains("Canon")) {
                    printerName = pName;
                }
            }
        }
    }
}

```

```
        break;
    }
}
// Print to the specified printer.
link.Print(printerName);
}
#endregion #printvialink
}
```

Visual Basic

```

(Form1.vb)
Imports System
Imports System.Windows.Forms
Imports System.Drawing.Printing
Imports DevExpress.XtraPrinting
Imports DevExpress.XtraRichEdit
Imports DevExpress.XtraRichEdit.API.Native
Namespace RichEdit_PrintingSystem
    Partial Public Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
        End Sub
        #Region "#printfromserver"
        Private Sub btn_PrintFromServer_Click(ByVal sender As Object, ByVal e As EventArgs)
            Dim srv As New RichEditDocumentServer()
            srv.LoadDocument("Grimm.docx", DocumentFormat.OpenXml)
            ' Insert a field displaying the current date/time into the document
            srv.BeginUpdate()
            Dim _header As SubDocument = srv.Document.Sections(0).BeginUpdateHeader()
            _header.Fields.Create(_header.Range.Start, "DATE \@ ""dddd, MMM dd, yyyy""")
            _header.Paragraphs(0).Alignment = ParagraphAlignment.Right
            srv.Document.Sections(0).EndUpdateHeader(_header)
            ' Specify page margins, orientation, etc.
            SetPrintOptions(srv)
            srv.EndUpdate()
            ' Display field values instead of code.
            srv.Options.MailMerge.ViewMergedData = True
            ' Create a printable link to print a document.
            PrintViaLink(srv)
        End Sub
        #EndRegion ' #printfromserver
        #Region "#setprintoptions"
        Private Shared Sub SetPrintOptions(ByVal richedit As IRichEditDocumentServer)
            For Each _section As Section In richedit.Document.Sections
                _section.Page.PaperKind = System.Drawing.Printing.PaperKind.Paper4
                _section.Page.Landscape = False
                _section.Margins.Left = 150F
                _section.Margins.Right = 150F
                _section.Margins.Top = 50F
                _section.Margins.Bottom = 50F
                _section.PageNumbering.NumberingFormat = NumberingFormat.C
                _section.PageNumbering.FirstPageNumber = 0
            Next _section
        End Sub
        #EndRegion ' #setprintoptions
        #Region "#printvialink"
        Private Shared Sub PrintViaLink(ByVal srv As RichEditDocumentServer)
            If Not srv.IsPrintingAvailable Then
                Return
            End If
        End Sub
    End Class
End Namespace

```

```
Dim link As New PrintableComponentLink(New PrintingSystem())
link.Component = srv
' Disable warnings.
link.PrintingSystem.ShowMarginsWarning = False
link.PrintingSystem.ShowPrintStatusDialog = False
' Find a printer containing 'Canon' in its name.
Dim printerName As String = String.Empty
For i As Integer = 0 To PrinterSettings.InstalledPrinters.Count - 1
    Dim pName As String = PrinterSettings.InstalledPrinters(i).Name
    If pName.Contains("Canon") Then
        printerName = pName
        Exit For
    End If
Next i
' Print to the specified printer.
link.Print(printerName)
End Sub
#EndRegion ' #printvialink
End Class
End Namespace
```

How to: Show a Print Preview Form

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Printing](#) > [How to: Show a Print Preview Form](#)

This example demonstrates how to invoke the Print Preview form using Word Processing Document API. To replicate this example, perform the following steps.

1. Create a RichEditDocumentServer instance in code.
2. Specify page settings for the first section in a document using the properties of the SectionPage object accessed via Section.Page.
3. Specify document content. In this example a simple multiplication table is inserted programmatically in a document.
4. Create a new PrintingSystem instance for creating and printing a document.
5. Create a PrintableComponentLink printing link with the specified printing system and assign the document server to the PrintableComponentLinkBase.Component property.
6. Call the LinkBase.CreateDocument method to generate a report and use the PrintingSystem.PreviewFormEx property to access the Print Preview form used to display a document preview. Call the PrintPreviewFormExBase.ShowDialog method to invoke the form.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E3190>.

The following code snippet creates a new document, modifies and prints it using the RichEditDocumentServer component.

Note

The code sample uses the PrintingSystem class. Make sure that the **DevExpress.XtraPrinting.v18.1.dll** assembly is added to your project.

C#

```
(Form1.cs)
using DevExpress.XtraRichEdit;
using DevExpress.XtraRichEdit.API.Native;
using DevExpress.XtraPrinting;
RichEditDocumentServer richServer = new RichEditDocumentServer();
// Specify default formatting
richServer.Document.DefaultParagraphProperties.Alignment = ParagraphAlignment.Center;
// Specify page settings
richServer.Document.Sections[0].Page.Landscape = true;
richServer.Document.Sections[0].Page.Height = DevExpress.Office.Utils.Units.InchesToDocumentsF(10.0f);
richServer.Document.Sections[0].Page.Width = DevExpress.Office.Utils.Units.InchesToDocumentsF(4.5f);
// Add document content
richServer.Document.AppendText("This content is created programmatically\n");
richServer.Document.AppendParagraph();
InsertTableIntoDocument(richServer);
// Invoke the Print Preview dialog
using (PrintingSystem printingSystem = new PrintingSystem()) {
    using (PrintableComponentLink link = new PrintableComponentLink(printingSystem)) {
        link.Component = richServer;
        link.CreateDocument();
        link.ShowPreviewDialog();
    }
}
```

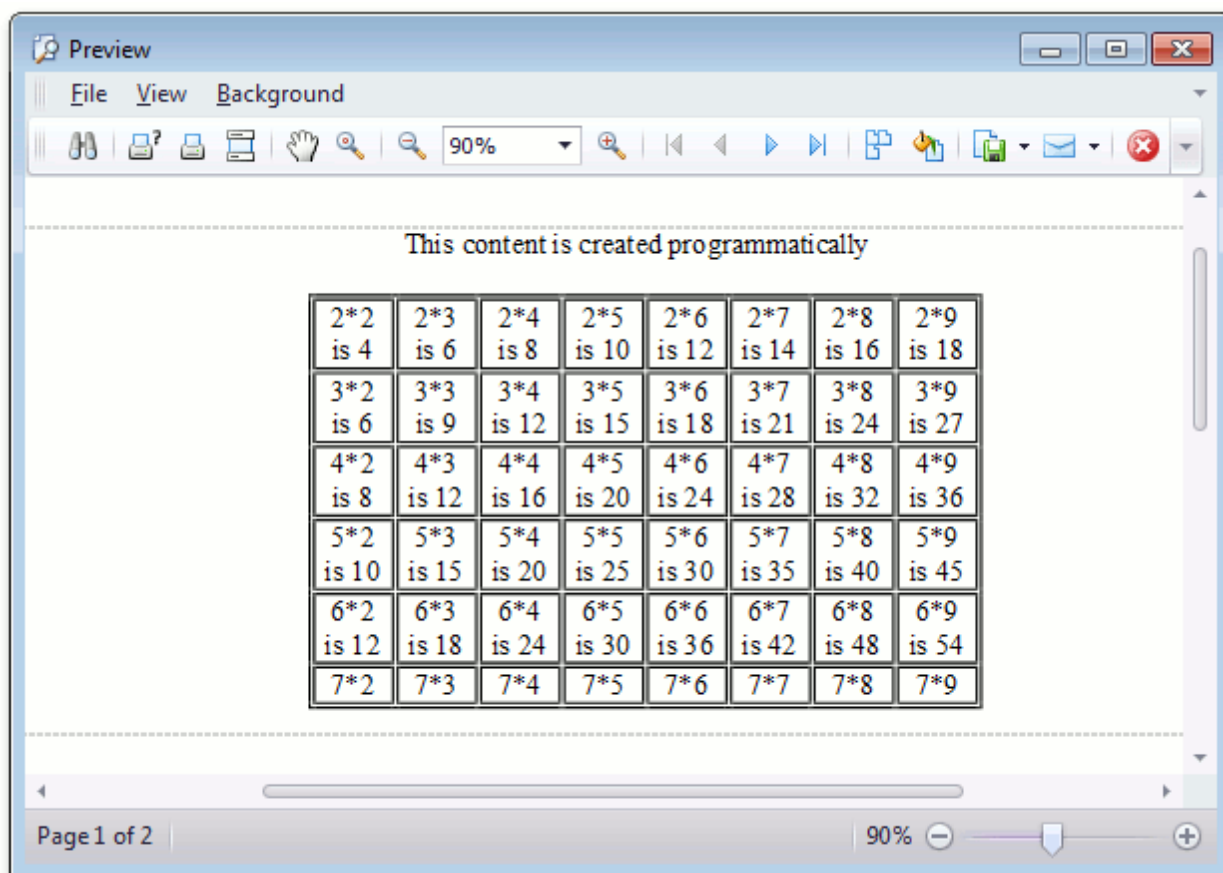
Visual Basic

```

(Form1.vb)
Imports DevExpress.XtraRichEdit
Imports DevExpress.XtraRichEdit.API.Native
Imports DevExpress.XtraPrinting
    Dim richServer As New RichEditDocumentServer()
    ' Specify default formatting
    richServer.Document.DefaultParagraphProperties.Alignment = ParagraphAl
    ' Specify page settings
    richServer.Document.Sections(0).Page.Landscape = True
    richServer.Document.Sections(0).Page.Height = DevExpress.Office.Utills.
    richServer.Document.Sections(0).Page.Width = DevExpress.Office.Utills.U
    ' Add document content
    richServer.Document.AppendText("This content is created programmatically")
    richServer.Document.AppendParagraph()
    InsertTableIntoDocument(richServer)
    ' Invoke the Print Preview dialog
    Using printingSystem As New PrintingSystem()
        Using link As New PrintableComponentLink(printingSystem)
            link.Component = richServer
            link.CreateDocument()
            link.ShowPreviewDialog()
        End Using
    End Using
End Using

```

The result is illustrated below.



Export

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Export](#)

This section contains the following examples:

- [How to: Save Selected Image to a File](#)
- [How to: Export Range To Different Formats](#)
- [How to: Determine Formatting Capabilities Required to Export the Document](#)
- [How to: Retain the Image URI in HTML Document](#)

How to: Save Selected Image to a File

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Export](#) > [How to: Save Selected Image to a File](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

To save an image from the document to a file, use the `ReadOnlyDocumentImageCollection.Get` method to obtain a collection of images within the specified range, and use the `OfficeImage.NativeImage` property, which allows access to a native **System.Drawing.Image** object. This object has the **Save** method that accomplishes the task.

C#

```
(InlinePictureActions.cs)
Document document = server.Document;
document.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml);
DocumentRange myRange = document.CreateRange(0, 100);
ReadOnlyDocumentImageCollection images = document.Images.Get(myRange);
if (images.Count > 0)
{
    DevExpress.Office.Utils.OfficeImage myImage = images[0].Image;
    System.Drawing.Image image = myImage.NativeImage;
    string imageName = String.Format("Image_at_pos_{0}.png", images[0].Range.Start.ToInt());
    image.Save(imageName);
    System.Diagnostics.Process.Start("explorer.exe", "/select," + imageName);
}
```

Visual Basic

```
(InlinePictureActions.vb)
Dim document As Document = server.Document
document.LoadDocument("Documents\\MovieRentals.docx", DocumentFormat.OpenXml)
Dim myRange As DocumentRange = document.CreateRange(0, 100)
Dim images As ReadOnlyDocumentImageCollection = document.Images.Get(myRange)
If images.Count > 0 Then
    Dim myImage As DevExpress.Office.Utils.OfficeImage = images(0).Image
    Dim image As System.Drawing.Image = myImage.NativeImage
    Dim imageName As String = String.Format("Image_at_pos_{0}.png", images(0).Range.Start.ToInt())
    image.Save(imageName)
    System.Diagnostics.Process.Start("explorer.exe", "/select," & imageName)
End If
```

How to: Export Range To Different Formats

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Export](#) > [How to: Export Range To Different Formats](#)

The following example describes how to export document range to different formats.

RcihEditDocumentServer provides a number of methods, allowing you to retrieve contents of the specified range in different formats. These are:

- SubDocument.GetText
- SubDocument.GetRtfText
- SubDocument.GetHtmlText
- SubDocument.GetMhtText
- SubDocument.GetWordMLText
- SubDocument.GetOpenXmlBytes.

To retrieve the document range into plane text, use the SubDocument.GetText, as illustrated below.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ExportActions.cs)
DevExpress.XtraRichEdit.API.Native.Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
string plainText = document.GetText(document.Paragraphs[2].Range);
System.Windows.Forms.MessageBox.Show(plainText);
```

Visual Basic

```
(ExportActions.vb)
Dim document As DevExpress.XtraRichEdit.API.Native.Document = server.Docum
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
Dim plainText As String = document.GetText(document.Paragraphs(2).Range)
System.Windows.Forms.MessageBox.Show(plainText)
```

The SubDocument.GetHtmlText is used to export range contents into HTML format. The following code demonstrates how to retrieve range into HTML format and open the resulting document in the browser windows:

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T418535>.

C#

```
(ExportActions.cs)
DevExpress.XtraRichEdit.API.Native.Document document = server.Document;
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml);
// Get the range for three paragraphs.
DocumentRange r = document.CreateRange(document.Paragraphs[0].Range.Start,
// Export to HTML.
string htmlText = document.GetHtmlText(r, null);
System.IO.File.WriteAllText("test.html", htmlText);
// Show the result in a browser window.
System.Diagnostics.Process.Start("test.html");
```

Visual Basic

```
(ExportActions.vb)
Dim document As DevExpress.XtraRichEdit.API.Native.Document = server.Docum
document.LoadDocument("Documents\\Grimm.docx", DocumentFormat.OpenXml)
' Get the range for three paragraphs.
Dim r As DocumentRange = document.CreateRange(document.Paragraphs(0).Range
' Export to HTML.
Dim htmlText As String = document.GetHtmlText(r, Nothing)
System.IO.File.WriteAllText("test.html", htmlText)
' Show the result in a browser window.
System.Diagnostics.Process.Start("test.html")
```

How to: Determine Formatting Capabilities Required to Export the Document

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Export](#) > [How to: Determine Formatting Capabilities Required to Export the Document](#)

This example illustrates how to check if specific formatting features are contained in the document loaded in the RichEditDocumentServer, and determine whether an exporter can perform the document export correctly.

The Document.RequiredExportCapabilities property provides access to an object which has several properties, indicating whether a particular formatting feature is used.

The following code snippet illustrates how you can detect if the capabilities of your custom exporter match the formatting capabilities required to export the current document. First, declare the exporter capabilities in the DocumentExportCapabilities structure, and then compare it with the structure obtained via the Document.RequiredExportCapabilities property.

C#

```
DevExpress.XtraRichEdit.DocumentExportCapabilities myExportFeatures =  
    new DevExpress.XtraRichEdit.DocumentExportCapabilities();  
myExportFeatures.CharacterFormatting = true;  
myExportFeatures.ParagraphFormatting = true;  
myExportFeatures.InlinePictures = true;  
if (myExportFeatures.Contains(richEditControl1.Document.RequiredExportCapabilities))  
    MessageBox.Show("The document can be exported");
```

Visual Basic

```
Dim myExportFeatures As DevExpress.XtraRichEdit.DocumentExportCapabilities =  
New DevExpress.XtraRichEdit.DocumentExportCapabilities()  
myExportFeatures.CharacterFormatting = True  
myExportFeatures.ParagraphFormatting = True  
myExportFeatures.InlinePictures = True  
If myExportFeatures.Contains(richEditControl1.Document.RequiredExportCapabilities) Then  
    MessageBox.Show("The document can be exported")  
End If
```

How to: Retain the Image URI in HTML Document

[Office File API](#) > [Word Processing Document API](#) > [Examples](#) > [Export](#) > [How to: Retain the Image URI in HTML Document](#)

The RichEditDocumentServer transforms an imported HTML document into a native document model and exports it from the native format to the resulting document's format when it is saved.

The images contained in the document can be saved in one of the following ways when saving a document as HTML:

- Embed images into the resulting file as base64-encoded data.

C#

```

```

- Save images as separate files. These images are saved as "image" followed by its sequential number to a folder with a name that combines the file name, an underscore and the word "files". For example, the first png image in the resulting Test.html file is saved to the Test_files\image0.png location.

C#

```


```

The table below describes the resulting document using different methods and properties:

Export	Result
RichEditControl.HtmlText property	Images are contained in the resulting file as base64-encoded data.
SubDocument.GetHtmlText method	The HtmlDocumentExporterOptions parameter allows specifying embedded or standalone images by setting the HtmlExportOptionsBase.EmbedImagesInHTML property.
RichEditControl.SaveDocument or RichEditControl.SaveDocumentAs methods, the SaveDocumentAsCommand command	The HtmlDocumentExporterOptions.EmbedImages property (accessible using the RichEditControl.ExportOptions.HtmlOptions.EmbedImages notation) determines whether images are embedded or saved as separate files.


The HTML exporter uses a URI provider instance to obtain each image's URI string. The built-in URI provider service (a service that implements the [IUriProviderService](#) interface) is used to register custom service providers of different types. Access the necessary provider by calling the [RichEditDocumentServer.GetService](#) method.

You can implement a custom URI provider to specify each image's location when the document is saved to an HTML format by creating an [IUriProvider](#) descendant and registering it in the [RichEditDocumentServer](#) services.

 **Tip**

The [SubDocument.GetHtmlText](#) method allows you to specify a custom URI provider as the method's parameter, bypassing the URI provider service and the [RichEditDocumentServer.ExportOptions.HtmlOptions.EmbedImages](#) option.

Each image in [RichEditDocumentServer](#) has an [OfficeImage.Uri](#) property that is the original image URI. The custom service's **CreateImageUri** method returns the [OfficeImage.Uri](#) value used to save the original image URI to the exported HTML document.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T605843>.

C#

```
(CustomUriProvider.cs)
using DevExpress.Office.Services;
using DevExpress.Office.Utils;
using System;

public class CustomUriProvider : IUriProvider
{
    public string CreateCssUri(string rootUri, string styleText, string relativeUri)
    {
        return String.Empty;
    }
    public string CreateImageUri(string rootUri, OfficeImage image, string relativeUri)
    {
        return image.Uri;
    }
}
```

C#

```
(Form1.cs)
private void server_DocumentLoaded(object sender, EventArgs e)
{
    IUriProviderService service = server.GetService<IUriProviderService>()
    if (service != null)
    {
        service.RegisterProvider(new CustomUriProvider());
    }
}
```

Visual Basic

```
(CustomUriProvider.vb)
Imports DevExpress.Office.Services
Imports DevExpress.Office.Utils
Imports System

Public Class CustomUriProvider
    Implements IUriProvider
    Private Function IUriProvider_CreateImageUri(rootUri As String, im
        Return image.Uri
    End Function
    Private Function IUriProvider_CreateCssUri(rootUri As String, styl
        Return String.Empty
    End Function
End Class
```

Visual Basic

```
(Form1.vb)
Private Sub server_DocumentLoaded(ByVal sender As Object, ByVal e As Event
    Dim service As IUriProviderService = server.GetService(Of IUriProvider
    If service IsNot Nothing Then
        service.RegisterProvider(New CustomUriProvider())
    End If
End Sub
```

See Also

[Import and Export](#)

PDF Document API

[Office File API](#) > [PDF Document API](#)

The PDF Document API is a non-visual .NET library that allows you to easily generate PDF files from scratch and manipulate existing PDF documents.

Getting Started	
<div><p>In this section, you will get acquainted with the PDF Document API.</p><ul style="list-style-type: none">Getting Started Describes the first steps to create a PDF processing console application in Visual Studio.Online Demos Shows all available demos related to DevExpress PDF Document API.Examples Contains task-based examples for the PDF Document API.</div>	<div></div>
Coordinate Systems	
<div><p>The topic in this section:</p><ul style="list-style-type: none">Coordinate Systems Describes coordinate systems used in the PDF Document API.</div>	
Generating a Document	
<div><p>The topic in this section:</p><ul style="list-style-type: none">Generating a Document Describes how to generate a document layout from scratch.</div>	
PDF Graphics	
<div><p>This section contains the following topics:</p><ul style="list-style-type: none">Overview Contains basic information on PDF Graphics API.Creating PDF Graphics Context Create PDF Graphics context.Drawing into Graphics Context Draw your own graphics into PDF graphics context using the following elements:</div>	

- Image
- String
- Graphics (lines, Bezier curves, polygons, ellipses and paths).
- [Adding Graphics Content to a Page](#)
Add created graphics content to a page.
- [Adding Interactive Form Fields](#)
Add interactive form fields to a document.

Document Manipulation

- This section contains all the required information related to programmatic document manipulation.
- [Merging Documents](#)
Merge different documents into a single PDF.
 - [Page Manipulation](#)
Add/Insert a new page into a document, copy pages from one document to another PDF, delete or rotate a page in a document.

Additional Content

- The topics in this section.
- [Bookmarks](#)
Create bookmarks from scratch, change bookmark attributes (e.g. rename a bookmark, change its destination or font style) and delete bookmarks.
 - [Hyperlinks](#)
Add a hyperlink to a page or URI.
 - [Attachments](#)
Attach any number of files to the PDF document and access attached files.

Interactive Forms

- This section contains all the required information related to an interactive form in a document.
- [Interactive Form Filling](#)
Fill existing interactive forms.

- [Interactive Form Flattening](#)
Flatten interactive forms in a document.
- [Creating Interactive Form](#)
Create interactive form fields and add them to a document.
- [Deleting Interactive Form](#)
Remove interactive form fields from a document.
- [Export and Import Interactive Form Data](#)
Export and import interactive form data in various formats - FDF, XFDF, XML and TXT.

Text Markup Annotations

The PDF Document API component provides API that can be used to manipulate (create, modify or remove) text markup annotations in pages.

The topics in this section contain all the required information related to text markup annotation manipulation in a document.

- [Creating](#)
Create markup annotations for a text and add them to a page.
- [Modifying](#)
Change existing text markup annotations in a page.
- [Deleting](#)
Remove text markup annotations from a page in a document.

Document Security

This section describes how to use save options to protect a document with a password and sign the document with a digital signature.

- [Protecting a Document](#)
Protect a document using both the owner and user passwords. In addition, you can optionally specify user permissions for printing, data extraction, document modification and interactive operations.
- [Signing a Document](#)
Apply a digital signature to a document.

Content Extraction

This section describes how to get content (an image, text) from a document.

- [Extracting Text from a Document](#)
Extract text from documents.
- [Extracting Images from a Document](#)
Extract images from documents.
- [Searching for Text in a Document](#)
Find text in documents.

Printing

This section describes how to print a document and customize print settings.

- [Printing](#)
Print any page in a document.

Export a Document to a Multi-Page Tiff Image


This section describes how to export document pages to a multi-page tiff image.

- [Export a PDF Document to a Multi-Page Tiff](#)
Export pages to a multi-page tiff image.

Getting Started

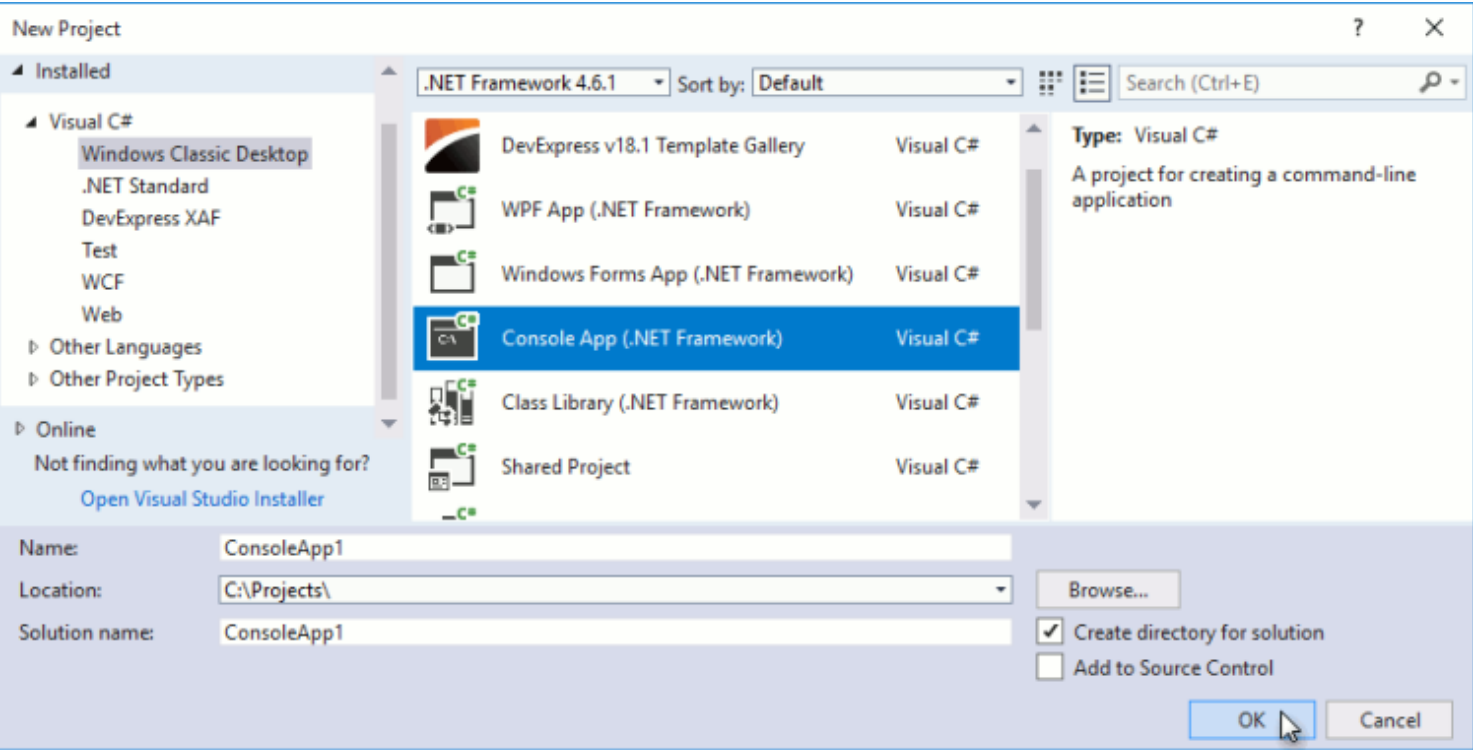
[Office File API](#) > [PDF Document API](#) > [Getting Started](#)

This tutorial describes the first steps to create a PDF processing console application in Visual Studio by using the non-visual PdfDocumentProcessor component.

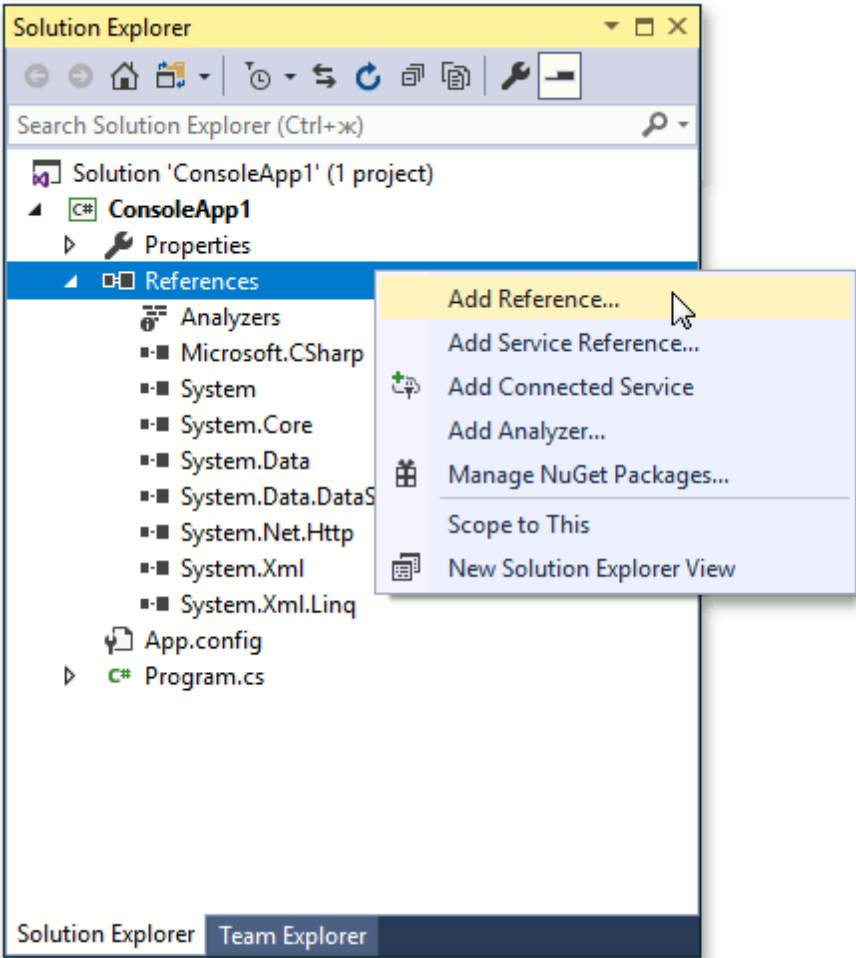
 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

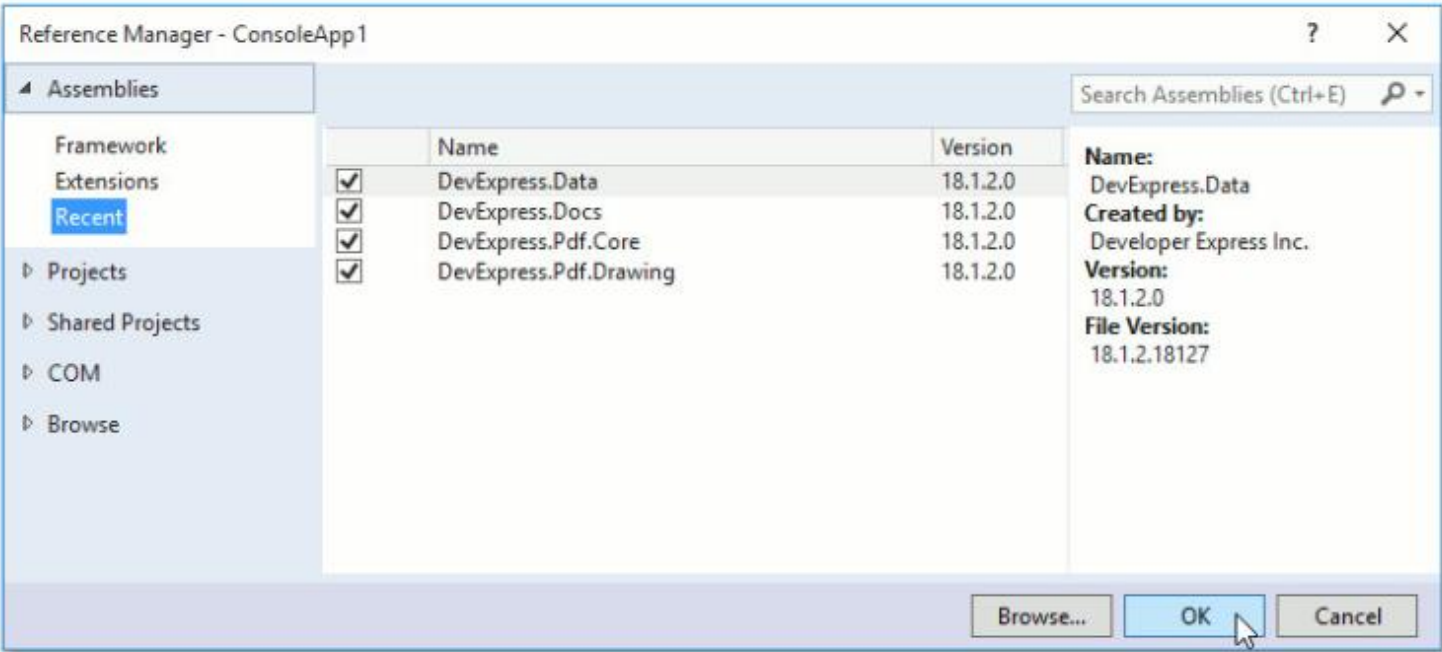
- To create a PDF Document API application, perform the following steps.
1. Run Microsoft Visual Studio 2010, 2012, 2013, 2015 or 2017.
 2. Start a new project (CTRL+SHIFT+N) and create a new **Console Application**. Specify the name of the project, and click **OK**.



3. Right-click **References** in the **Solution Explorer** and select **Add Reference...** in the invoked context menu.



4. Select the **DevExpress.Data.v18.1**, **DevExpress.Docs.v18.1**, **DevExpress.Pdf.v18.1.Core**, and **DevExpress.Pdf.v18.1.Drawing** assemblies and click **OK**.



5. Perform the necessary document processing in the **Main** method of the application's **Program.cs** file (**Main** procedure of the **Module1.vb** file for Visual Basic).

For example, the following code opens a sample document, rotates its pages and saves the modified document to a new location.

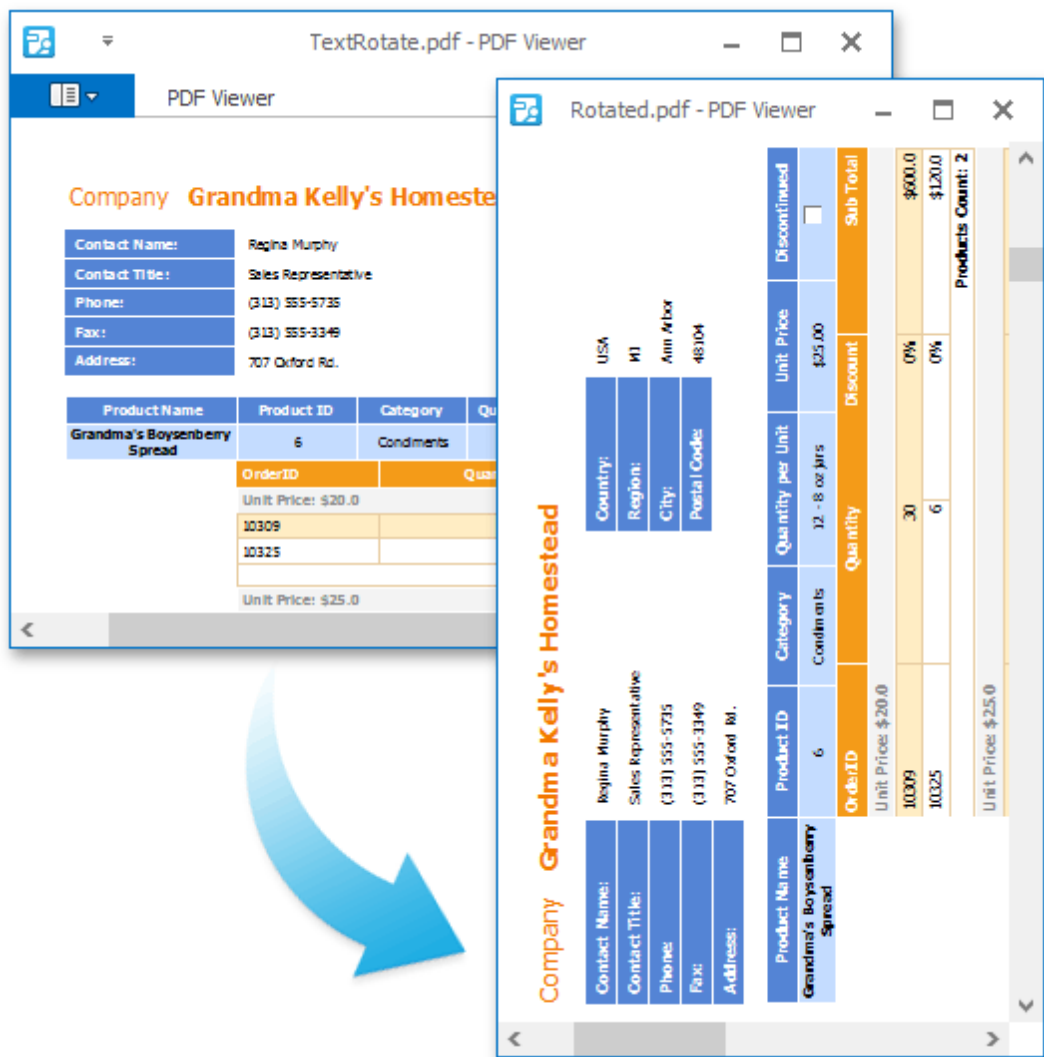
Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T114305>.

This example illustrates how to use the PDF Document API component for rotating pages.

C#	
<pre>(Program.cs) using DevExpress.Pdf; namespace PdfPageRotationExample { class Program { static void Main(string[] args) { using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) { pdfDocumentProcessor.LoadDocument("..\\..\\docs\\TextRotate.pdf"); int angle = 0; foreach (PdfPage page in pdfDocumentProcessor.Document.Pages) { angle = (angle + 90) % 360; page.Rotate = angle; } pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Rotated.pdf"); } } } }</pre>	
Visual Basic	
<pre>(Program.vb) Imports DevExpress.Pdf Namespace PdfPageRotationExample Friend Class Program Shared Sub Main(ByVal args() As String) Using pdfDocumentProcessor As New PdfDocumentProcessor() pdfDocumentProcessor.LoadDocument("..\\..\\docs\\TextRotate.p Dim angle As Integer = 0 For Each page As PdfPage In pdfDocumentProcessor.Document. angle = (angle + 90) Mod 360 page.Rotate = angle Next page pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Rotated.pdf" End Using End Sub End Class End Namespace</pre>	<pre> pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Rotated.pdf"); } } } }</pre>

Run the project. After executing the code above, the pages are rotated and the new document is saved to the specified location.



See Also
[Generating a Document](#)
[Additional Content](#)
[Protecting a Document](#)
[Signing a Document](#)

Coordinate Systems

[Office File API](#) > [PDF Document API](#) > [Coordinate Systems](#)

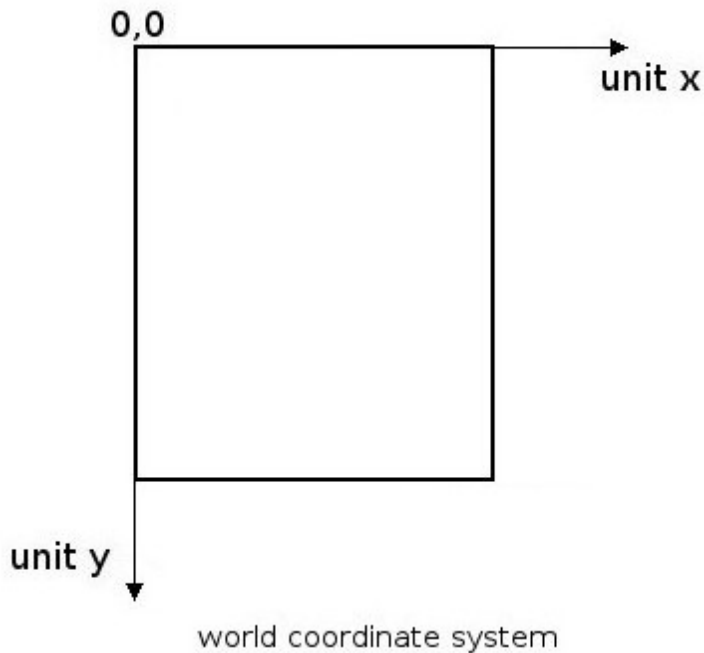
The coordinate systems define the position, orientation, and size of text, graphics, and images that appear on a page. The coordinate systems are determined with respect to the current page.

This document describes coordinate systems used in the [PDF Document API](#) and how transformations among these coordinate systems are specified.

- [World Coordinate System](#)
- [User Coordinate system](#)
- [Page Coordinate System](#)
- [Using Coordinate Systems in API](#)
- [Coordinate Transformations](#)
- [Converting World Coordinates to Page Coordinates](#)
- [Converting Page Coordinates to World Coordinates](#)

World Coordinate System

The world coordinate system models a particular graphics world. The world coordinate system's origin (0,0) is at the top left of the page. A positive x increases towards the right and a positive y towards the bottom.

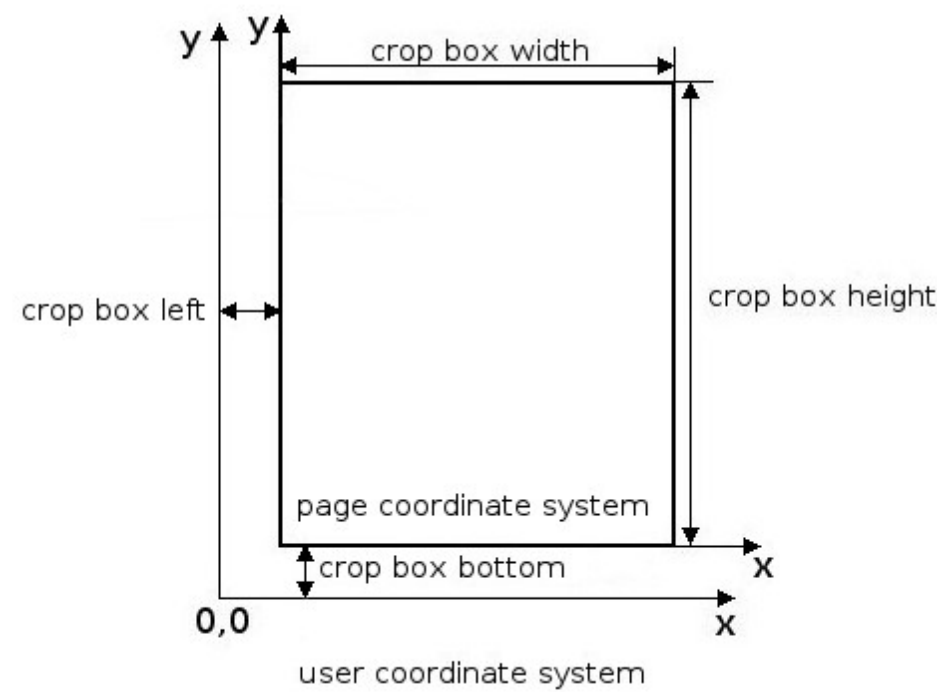


To learn about API that uses the world coordinate system, see [Using Coordinate Systems in API](#).

User Coordinate system

The positive x axis extends horizontally to the right and the positive y axis vertically upward. The page boundaries are defined by the crop box in the user coordinate system. The crop box designates the visible page area that is displayed or printed. To get the page crop box, use the PdfPageTreeObject.CropBox property. The size of the unit in the user space is a point (1/72 of an inch).

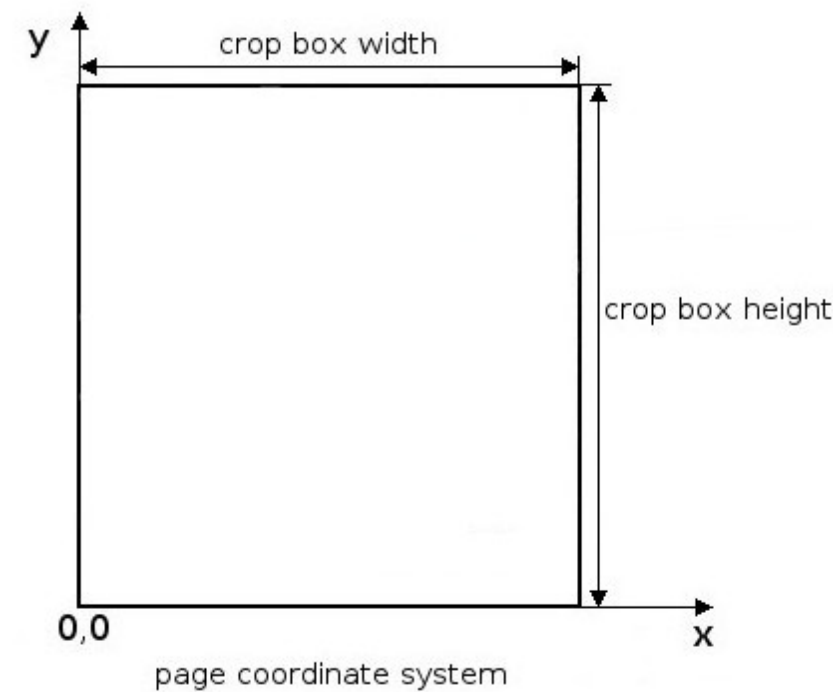
The image below shows the conversion from the user coordinate system to the page coordinate system.



The user coordinate system is an internal PDF coordinate system. The document processor's low-level content API (document model) uses the PDF user coordinate system. A PdfDocument object that can be accessed using the PdfDocumentProcessor.Document property represents the document model. The high-level API refers to the page coordinate system. So, you do not need to use the user coordinate system in most cases.

Page Coordinate System

The page coordinate system was introduced to simplify the page calculations. The page coordinate system's origin $(0, 0)$ is shifted by page crop bottom and page crop box left values relative to the user coordinate system's origin (see the image above). The lower-left corner of the page coincides with the origin of the page coordinate system $(0, 0)$. The Y axis is directed from the bottom of the page to the top.



To learn about API that uses the page coordinate system, see [Using Coordinates Systems in API](#).

Using Coordinate Systems in API

The PdfGraphics class (see the [PDF Graphics](#) section for details) uses the [world coordinate system](#).

The following API uses the [page coordinate system](#):

PDF Document API

- overloaded [PdfDocumentProcessor.FindText](#) methods
- overloaded [PdfDocumentProcessor.GetImages](#) methods
- overloaded [PdfDocumentProcessor.GetText](#) methods
- the PdfWord class.

WinForms PDF Viewer

- overloaded PdfViewer.FindText methods.

WPF PDF Viewer

- the PdfViewerControl.FindText method
- overloaded PdfViewerControl.GetText methods
- overloaded PdfViewerControl.ScrollIntoView methods

Coordinate Transformations

You need to [convert page coordinates to world coordinates](#), for example, to highlight search results in a document using the PDF Document API (see [How to: Highlight Search Results in a Document](#) for more details). The search results are obtained from the overloaded [PdfDocumentProcessor.FindText](#) method in the page coordinates. To draw filled rectangles that correspond to the document area containing found text, convert **page coordinates to world coordinates** since the PdfGraphics.FillRectangle method uses world coordinates.

Unlike the page coordinates to world coordinates transformation, the [world coordinates are transformed to page coordinates](#) automatically using the following PDF Document API's methods:

- overloaded [PdfDocumentProcessor.CreateDestination](#) methods
- overloaded [PdfDocumentProcessor.RenderNewPage](#) methods (see [Adding Graphics Content to a Page](#))
- overloaded PdfGraphics.AddToPageBackground methods (see [Adding Graphics Content to a Page](#))
- overloaded PdfGraphics.AddToPageForeground methods (see [Adding Graphics Content to a Page](#))

Converting World Coordinates to Page Coordinates

To convert a graphics unit into a physical value such as inches, we should provide a resolution's value (dpi - the number of dots (units) per inch). The dpi shows how many units from the world coordinates are placed in an inch on a page. So, to get the value in inches, we should divide units by dpi values (for example, 400 units divided by 200 dpi equals 2 inches). The measurement unit in the page coordinate system is a point (1/72 of an inch).

The world coordinates (units) are converted to page coordinates (points) using the following formula:

$$x = (unitX / dpiX) * 72;$$

$$y = cropBoxHeight - (unitY / dpiY) * 72.$$

Use the PdfPageTreeObject.CropBox property to get the page crop box height.

Example

Two points represent a rectangle in the world coordinates. The first point is (0, 0), the second point is (400, 600) units and the dpi value is 200.

Using the formula above, we get the following result in the page coordinates:

The first point:

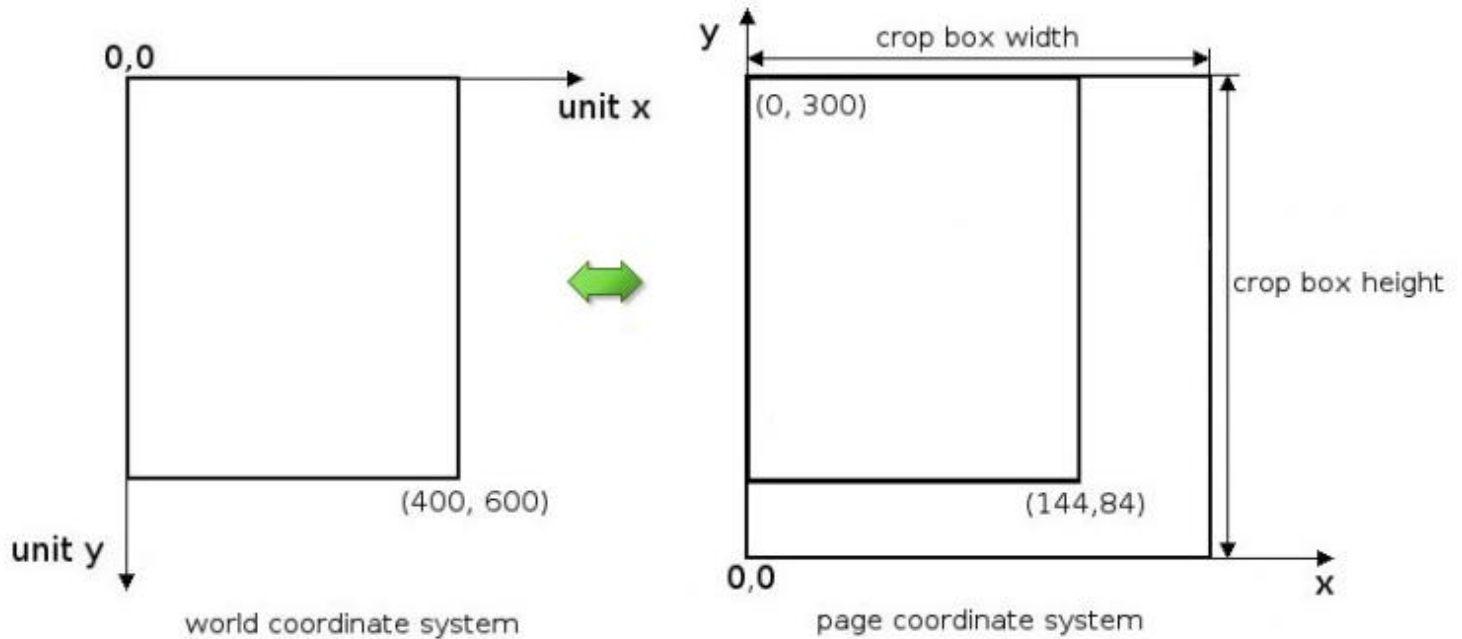
$$\text{Point X1 value} = (0 / 200) * 72 = 0$$

$$\text{Point Y1 value} = 300 - (0 / 200) * 72 = 300 \text{ (equal to crop box height)}$$

The second point:

$$\text{Point X2 value} = (400 / 200) * 72 = 144$$

Point Y2 value = $(300 - (600 / 200) * 72) = 84$



Converting Page Coordinates to World Coordinates

The page coordinates (points) are converted to world coordinates (units) using the following formula:

$$\text{unitX} = (x / 72) * \text{dpiX};$$

$$\text{unitY} = ((\text{cropBoxHeight} - y) / 72) * \text{dpiY}.$$

Example

Let's take a rectangle represented by two points obtained from the previous example (see the section above). The first point is (0, 300), the second point is (144, 84) and the dpi value is 200.

Using the formula above, we get the following result:

The first point in the world coordinates:

$$\text{unitX1} = (0 / 72) * 200 = 0;$$

$$\text{unitY1} = ((300 - 300) / 72) * 200 = 0.$$

The second point in the world coordinates:

$$\text{unitX2} = (144 / 72) * 200 = 400;$$

$$\text{unitY2} = ((300 - 84) / 72) * 200 = 600.$$

Generating a Document

[Office File API](#) > [PDF Document API](#) > [Generating a Document](#)

This topic describes how to generate a document layout from scratch.

Firstly, create a document that has no pages by creating a [PdfDocumentProcessor](#) object and calling one of the [PdfDocumentProcessor.CreateEmptyDocument](#) overload methods.

These methods can be called using a stream, a file path (see the code snippet below), creation options (represented by an instance of the PdfCreationOptions class) or save options (represented by an instance of the PdfSaveOptions class).

Populate a document with graphic content:

- Draw graphics on a page by creating a PdfGraphics object using the [PdfDocumentProcessor.CreateGraphics](#) method and calling the **Draw** method for corresponding elements (for example, the PdfGraphics.DrawString overload method draws a text string at the specified location with the specified **SolidBrush** and **Font** objects).

Note

You need to reference the **DevExpress.Pdf.Drawing** assembly to draw graphic content on a page because it requires an instance of the PdfGraphics class.

- Render a page with created graphics by calling the [PdfDocumentProcessor.RenderNewPage](#) overload method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

C#

```
void DoSomething() {
    using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
        processor.CreateEmptyDocument("../\\..\\Document.pdf");
        using (PdfGraphics graph = processor.CreateGraphics()) {
            DrawGraphics(graph);
            processor.RenderNewPage(PdfPaperSize.Letter, graph);
        }
    }
}

private void DrawGraphics(PdfGraphics graph) {
    SolidBrush black = (SolidBrush)Brushes.Black;
    using (Font font = new Font("Times New Roman", 32, FontStyle.Bold)) {
        graph.DrawString("PDF Document Processor", font, black, 180, 150);
    }
}
```

Visual Basic

```
Private Sub DoSomething()
    Using processor As New PdfDocumentProcessor()
        processor.CreateEmptyDocument("../\\..\\Document.pdf")
        Using graph As PdfGraphics = processor.CreateGraphics()
            DrawGraphics(graph)
            processor.RenderNewPage(PdfPaperSize.Letter, graph)
        End Using
    End Using
End Sub

Private Sub DrawGraphics(graph As PdfGraphics)
    Dim black As SolidBrush = DirectCast(Brushes.Black, SolidBrush)
    Using font As New Font("Times New Roman", 32, FontStyle.Bold)
        graph.DrawString("PDF Document Processor", font, black, 180, 150)
    End Using
End Sub
```

See the [PDF Graphics](#) topics for more information about PDF Graphics.

The PDF Document API provides additional settings to customize document generation. See the sections below.

Font Embedding

The PDF Document API component creates a document with embedded fonts by default.

To prohibit embedding all fonts, set the PdfCreationOptions.DisableEmbeddingAllFonts property to **true** and pass a PdfCreationOptions object containing this setting as a parameter to one of the [PdfDocumentProcessor.CreateEmptyDocument](#) overload methods.

Use the PdfCreationOptions.NotEmbeddedFontFamilies property to specify which font families should not be embedded in a document.

Specifying a Document's Compatibility Mode

The following compatibility modes are supported:

- PDF (ISO 32000-1:2008) - default mode
- PDF/A-1b (ISO 19005-1)
- PDF/A-2b (ISO 19005-2:2011)
- PDF/A-3b (ISO 19005-3:2012)

To create a PDF/A-compatible document, set the PdfCreationOptions.Compatibility property to either PdfCompatibility.PdfA1b, PdfCompatibility.PdfA2b or PdfCompatibility.PdfA3b. Then, call the [PdfDocumentProcessor.CreateEmptyDocument](#) overload method and pass a PdfCreationOptions object containing this setting as a parameter.

```
C#  
  
using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {  
    processor.CreateEmptyDocument("../\\..\\Result.pdf", new PdfCreationOptions() {  
        Compatibility = PdfCompatibility.PdfA2b  
    });  
}
```

```
Visual Basic  
  
Imports (PdfDocumentProcessor processor = New PdfDocumentProcessor()) {  
    Private Function PdfCreationOptions() As processor.CreateEmptyDocument("../\\..\\Result.pdf", Shadows  
        Compatibility = PdfCompatibility.PdfA2b  
    End Function  
)  
}
```

PDF/A has the following limitations:

- Non-embedded fonts are not supported;
- PDF/A-1b and PDF/A-2b documents cannot contain file attachments;
- Encryption is forbidden;
- Transparency is not allowed in a PDF/A-1b document (all transparency information is ignored, and no exceptions are raised).

Specifying Encryption Settings and Signature

The PDF Document API can protect a document with user and owner passwords. These passwords are used to prevent users from accessing or modifying PDF documents. All required settings to protect a document are contained in the PdfEncryptionOptions object, which can be accessed via the PdfSaveOptions.EncryptionOptions property. To encrypt the document with these settings, call the [PdfDocumentProcessor.CreateEmptyDocument](#) overload method and pass the PdfSaveOptions object containing these settings as a parameter. See the [Protecting a Document](#) topic for more information.

To specify a signature, use the PdfSaveOptions.Signature property. See the [Signing a Document](#) topic to learn more.

See Also

[How to: Generate a Document Layout from Scratch](#)
[How to: Protect a PDF Document with a Password](#)

PDF Graphics

[Office File API](#) > [PDF Document API](#) > [PDF Graphics](#)

This section contains the following topics:

- [Overview](#)
Contains basic information on PDF Graphics API.
- [Creating PDF Graphics Context](#)
Describes how to create PDF Graphics context.
- [Drawing into Graphics Context](#)
Describes how to draw your own graphics into PDF graphics context using the following elements:
 - Image
 - String
 - Graphics (lines, Bezier curves, polygons, ellipses and paths).
- [Adding Graphics Content to a Page](#)
Explains how to add created graphics content to a page.
- [Adding Interactive Form Fields](#)
Explains how to add interactive form fields to a document.

Overview

[Office File API](#) > [PDF Document API](#) > [PDF Graphics](#) > [Overview](#)

The PDF graphics context that allow an application to draw on a page is represented by an instance of the PdfGraphics class. This class provides an API similar to the standard **System.Drawing.Graphics** class's API - you can draw on graphics context the same way as you do using the GDI+ API.

In addition to methods used to create an image (PdfGraphics.DrawImage), text (PdfGraphics.DrawString) and graphic elements in a document (e.g, PdfGraphics.DrawBezier), the PdfGraphics class also provides the following properties:

Member	Description
PdfGraphics.ConvertImagesToJpeg	Specifies whether to convert bitmap images to the Jpeg format reducing the size of the resulting PDF document.
PdfGraphics.JpegImageQuality	Gets or sets the quality of Jpeg images in the resulting PDF file.
PdfGraphics.TextOrigin	Specifies how to interpret a point passed to one of the DrawString overload methods that take a PointF object.
PdfGraphics.UseKerning	Gets or sets a value which indicates whether kerning is used when drawing characters.

The **PdfGraphics** class uses a world coordinate system that models a particular graphics world. See the [Coordinate Systems](#) topic to learn more.

If you have an existing document and want to draw graphics content at the required position of the page, you need to convert page coordinates to world coordinates.

To transform page space coordinates to world coordinates:

- Obtain a world Y coordinate by subtracting the page space Y value from the page crop box height, the X coordinate remains the same.
- Draw graphics content (e.g. a text line) on the page by calling **Draw** methods of the PdfGraphics class using the converted point.

To add graphics content to a page, transform world coordinates to page space coordinates by calling one of the PdfGraphics.AddToPageForeground, PdfGraphics.AddToPageBackground or [PdfDocumentProcessor.RenderNewPage](#) overload methods. Since page unit of measurement is a point (1/72 of an inch), pass 72 as **dpiX** and **dpiY** values to one of these methods. See [Adding Graphics Content to a Page](#) for more details.

Example

This example shows how to draw graphics content on an existing page using the transformation from the page to world coordinates described above (see [Coordinate Systems](#) for more details).

Before drawing shapes, lines and other graphic content on a page, you need to create a PdfGraphics object. See [Creating PDF Graphics Context](#) for more information on how to create **PdfGraphics**.

Use PdfGraphics class methods (for example, PdfGraphics.DrawImage, PdfGraphics.DrawString, PdfGraphics.DrawPolygon, PdfGraphics.DrawRectangle, etc.) to draw into a PDF graphics context. See [Drawing into Graphics Context](#) for more information.

After creating graphics context and drawing on it, you need to add graphics content to a page. See [Adding Graphics Content to a Page](#).

C#

```

using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
    // Load a document.
    processor.LoadDocument("../..\\Document.pdf");
    // Create graphics context.
    using (PdfGraphics graphics = processor.CreateGraphics()) {
        // Create a point in the page coordinate system.
        PdfPoint sourcePoint = new PdfPoint(10, 100);
        // Convert a source point from the page coordinate system
        // to world coordinate system.
        PointF convertedPoint = new PointF((float)sourcePoint.X,
            (float)(processor.Document.Pages[0].CropBox.Height
                - sourcePoint.Y));
        // Draw a text line on the page using the converted point.
        SolidBrush black = (SolidBrush)Brushes.Black;
        using (Font font1 =
            new Font("Times New Roman", 32, FontStyle.Bold)) {
            graphics.DrawString("PDF Document Processor", font1,
                black, convertedPoint);
        }
        // Add graphics content to the page foreground.
        graphics.AddToPageForeground(processor.Document.Pages[0],
            72, 72);
    }
    // Save the result document.
    processor.SaveDocument("../..\\Result.pdf");
}

```

Visual Basic

```

Using processor As New PdfDocumentProcessor()
    ' Load a document.
    processor.LoadDocument("../..\\Document.pdf")
    ' Create graphics context.
    Using graphics As PdfGraphics = processor.CreateGraphics()
        ' Create a point in the page coordinate system.
        Dim sourcePoint As New PdfPoint(10, 100)
        ' Convert a source point from the page coordinate system
        ' to world coordinate system.
        Dim convertedPoint As New PointF(CSng(sourcePoint.X),
            CSng(processor.Document.Pages(0).CropBox.Height - sourcePoint.Y))
        ' Draw a text line on the page using the converted point.
        Dim black As SolidBrush = DirectCast(Brushes.Black, SolidBrush)
        Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
            graphics.DrawString("PDF Document Processor", font1, black, convertedPoint)
        End Using
        ' Add graphics content to the page foreground.
        graphics.AddToPageForeground(processor.Document.Pages(0), 72, 72)
    End Using
    ' Save the result document.
    processor.SaveDocument("../..\\Result.pdf")
End Using

```

Creating PDF Graphics Context

[Office File API](#) > [PDF Document API](#) > [PDF Graphics](#) > [Creating PDF Graphics Context](#)

Before drawing shapes, lines and other graphics content on a page, you need to create PDF graphics context. The graphic context is represented by a PdfGraphics object.

Note

To access this class, reference the **DevExpress.Pdf.Drawing** assembly.

To create a PdfGraphics object, call the [PdfDocumentProcessor.CreateGraphics](#) method.

The PDF graphics context is associated with a document. This means that the PDF graphics context cannot be created without loading the document into the PDF Document API component. If you close the document, reopen it or open another document, the PDF graphics context becomes invalid and the **ArgumentException** is raised when the graphics context is applied to the new document. See [Adding Graphics Content to a Page](#) for more details.

Drawing into Graphics Context

[Office File API](#) > [PDF Document API](#) > [PDF Graphics](#) > [Drawing into Graphics Context](#)

The PdfGraphics class contains methods (GDI+ like) that allow you to draw different shapes, lines, and images into a PDF graphics context.

Member	Description
PdfGraphics.DrawLine	Draws a line connecting two points on the page specified by the coordinate pairs.
PdfGraphics.DrawLines	Draws a series of line segments that connect an array of PointF structures.
PdfGraphics.DrawBezier	Draws a Bezier spline defined by four Point structures.
PdfGraphics.DrawBeziers	Draws a series of Bezier splines from an array of PointF structures.
PdfGraphics.DrawEllipse	Draws an ellipse specified by a RectangleF structure.
PdfGraphics.DrawRectangle	Draws a rectangle specified by a Rectangle structure.
PdfGraphics.DrawPolygon	Draws a polygon defined by an array of PointF structures.
PdfGraphics.DrawPath	Draws a path on a page.
PdfGraphics.DrawString	A set of overloaded methods used to draw the text string with the specified SolidBrush, Font objects and other parameters.
PdfGraphics.DrawImage	A set of overloaded methods used to draw an image using the specified parameters.
PdfGraphics.FillRectangle	Fills the interior of a rectangle specified by a RectangleF structure.
PdfGraphics.FillPath	Fills the interior of a GraphicsPath.
PdfGraphics.FillEllipse	Fills the interior of an ellipse defined by a bounding rectangle specified by a RectangleF structure.
PdfGraphics.FillPolygon	Fills the interior of a polygon defined by an array of points specified by PointF structures.
PdfGraphics.RotateTransform	Rotates the coordinate system clockwise relative to its origin to the specified angle.
PdfGraphics.ScaleTransform	Scales the coordinate system according to the specified scale factor.
PdfGraphics.TranslateTransform	Translates the origin of the coordinate system to the specified point.

When an image is drawn on a page using one of the overloaded PdfGraphics.DrawImage methods, the PDF Document API may produce a resulting PDF file of the large size. To reduce the size of the PDF file, use the PdfGraphics.ConvertImagesToJpeg and PdfGraphics.JpegImageQuality properties.

To customize text drawing, use the PdfGraphics.TextOrigin and PdfGraphics.UseKerning properties before calling the overloaded PdfGraphics.DrawString methods.

Example

This example shows how to draw text lines on a page using the PdfGraphics.DrawString method.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

C#

```
static void DrawGraphics(PdfGraphics graphics) {
    SolidBrush black = (SolidBrush)Brushes.Black;
    using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
        graphics.DrawString("PDF Document Processor", font1, black, 180, 150);
    }
    using (Font font2 = new Font("Arial", 20)) {
        graphics.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
    }
    using (Font font3 = new Font("Arial", 10)) {
        graphics.DrawString("The PDF Document Processor is a non-visual component " +
            "that provides the application programming interface of the PDF Viewer.",
            font3, black, 180, 250);
    }
}
```

Visual Basic

```
Private Shared Sub DrawGraphics(graphics As PdfGraphics)
    Dim black As SolidBrush = DirectCast(Brushes.Black, SolidBrush)
    Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
        graphics.DrawString("PDF Document Processor", font1, black, 180, 150)
    End Using
    Using font2 As New Font("Arial", 20)
        graphics.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
    End Using
    Using font3 As New Font("Arial", 10)
        graphics.DrawString("The PDF Document Processor is a non-visual component " +
            "that provides the application programming interface of the PDF Viewer.", font3,
            black, 180, 250)
    End Using
End Sub
```

Adding Graphics Content to a Page

[Office File API](#) > [PDF Document API](#) > [PDF Graphics](#) > [Adding Graphics Content to a Page](#)

When adding graphics content to an entire page or a part of it, the world coordinates are converted to page coordinates (see [Coordinate Systems](#) to learn more).

The sections below describe methods used to add graphics content to a page.

Adding graphics content to an existing page

To add graphics content to a foreground/background of a page, call the overloaded PdfGraphics.AddToPageForeground or PdfGraphics.AddToPageBackground method and pass a PdfPage object containing the page where graphics should be drawn, and **dpiX** and **dpiY** values as arguments. Use the PdfDocument.Pages property to access a collection of pages within a document, and the [PdfDocumentProcessor.Document](#) property to get a document.

Adding graphics content to a new page

To render a new page with graphics content, call the overloaded [PdfDocumentProcessor.RenderNewPage](#) method and pass graphics content (represented by a PdfGraphics object), a PdfRectangle object containing the page size, and **dpiX** and **dpiY** values as arguments.

Note

The **dpiX** and **dpiY** values are equal to 96 in all overloaded [PdfDocumentProcessor.RenderNewPage](#), PdfGraphics.AddToPageForeground, PdfGraphics.AddToPageBackground methods. To render graphics content on a page at another dpi, pass **dpiX** and **dpiY** values to one of these overloaded methods.

A page's foreground content (text, an image) can overlap a watermark added to a page's background using the PdfGraphics.AddToPageBackground method. We recommend placing a semi-transparent watermark on the page's foreground (instead of the page's background) by calling the PdfGraphics.AddToPageForeground method.

See Also
[Overview](#)
[How to: Create Graphics in a Document with Landscape and Portrait Page Orientations](#)

Adding Interactive Form Fields

[Office File API](#) > [PDF Document API](#) > [PDF Graphics](#) > [Adding Interactive Form Fields](#)

You can create interactive form fields (for example, a text box, combo box, check box, list box) and add these fields to a document using graphics represented by an instance of the PdfGraphics class.

To create a PdfGraphics object, call the PdfDocumentProcessor.CreateGraphics method.

Note

To access the PdfGraphics class, you need to reference the **DevExpress.Pdf.Drawing** assembly.

The PdfGraphics class methods use a world coordinate system. See [Coordinate Systems](#) to learn more about this coordinate system.

An instance of the PdfGraphicsAcroFormField object represents the interactive form field.

To create a form field:

- create an instance of the class that represents the specific form field (for example, PdfGraphicsAcroFormTextBoxField);
- or
- call the corresponding static **Create** method for the PdfGraphicsAcroFormField class (for example, PdfGraphicsAcroFormField.CreateTextBox).

The PdfGraphicsAcroFormField.CreateTextBox method returns a PdfGraphicsAcroFormTextBoxField object that represents a created text box field. This method needs the name of the field and a **RectangleF** instance that defines the position of the form field on a document page.

Then, specify the interactive form field properties. For example, you can specify the text box text and type using PdfGraphicsAcroFormTextBoxField.Text, and PdfGraphicsAcroFormTextBoxField.Type properties, respectively.

To specify the text box name, tooltip and appearance settings, use the PdfGraphicsAcroFormField.Name, PdfGraphicsAcroFormField.ToolTip, and PdfGraphicsAcroFormField.Appearance properties.

Note

Form field names must be unique in an interactive form.

To add an interactive form field to graphics, call the PdfGraphics.AddFormField method, and pass the form field to this method as an argument.

The graphics that contain interactive form fields can be added only to a single page.

To clear form fields that were previously added to graphics, call the PdfGraphics.ClearFormFields method.

Finally, add interactive form fields to a document using one of the following ways:

- Render a new page with created interactive form fields by calling one of the PdfDocumentProcessor.RenderNewPage overloaded methods;
- Add the interactive form fields to an existing document page by calling one of the overloaded methods: PdfGraphics.AddToPageBackground, and PdfGraphics.AddToPageForeground.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494157>.

This example shows how to create a text box field and add it to a document.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System.Drawing;
namespace AddTextBoxField {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw a text box field.
                using (PdfGraphics graphics = processor.CreateGraphics()) {
                    DrawTextBoxField(graphics);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics);
                }
            }
        }
        static void DrawTextBoxField(PdfGraphics graphics) {
            // Create a text box field and specify its location on the page using a RectangleF object.
            PdfGraphicsAcroFormTextBoxField textBox = new PdfGraphicsAcroFormTextBoxField("text box", new
            // Specify text box properties.
            textBox.Text = "Text Box";
            textBox.TextAlignment = PdfAcroFormStringAlignment.Near;
            textBox.Appearance.FontSize = 12;
            textBox.Appearance.BackgroundColor = Color.AliceBlue;
            // Add the field to graphics.
            graphics.AddFormField(textBox);
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Drawing
Namespace AddTextBoxField
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw a text box field.
                Using graphics As PdfGraphics = processor.CreateGraphics()
                    DrawTextBoxField(graphics)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawTextBoxField(ByVal graphics As PdfGraphics)
            ' Create a text box field and specify its location on the page
            Dim textBox As New PdfGraphicsAcroFormTextBoxField("text box",
            ' Specify text box properties.
            textBox.Text = "Text Box"
            textBox.TextAlignment = PdfAcroFormStringAlignment.Near
            textBox.Appearance.FontSize = 12
            textBox.Appearance.BackgroundColor = Color.AliceBlue
            ' Add the field to graphics.
            graphics.AddFormField(textBox)
        End Sub
    End Class
End Namespace
```

Document Manipulation

[Office File API](#) > [PDF Document API](#) > [Document Manipulation](#)

This section contains all the required information related to programmatic document manipulation.

The topics in this section:

- [Merging Documents](#)
Merge different documents into a single PDF file.
- [Page Manipulation](#)
Add/Insert a new page into a document, copy pages from one document to another PDF, delete or rotate a page in a document.

Merging Documents

[Office File API](#) > [PDF Document API](#) > [Document Manipulation](#) > [Merging Documents](#)

The PDF Document API component can merge multiple PDF documents into a single PDF file.

This task can be achieved by creating a target document at the beginning of the merging process, and adding source documents to the target document one-by-one.

Note

The source documents are added to the end of a target document starting from the new page.
The PDF Document API component keeps original document content while merging documents.

To create a target document with no pages, create an instance of the PdfDocumentProcessor object and call one of the overloaded PdfDocumentProcessor.CreateEmptyDocument methods (e.g., using a file path).

To add a source document to the target document, call one of the overloaded PdfDocumentProcessor.AppendDocument methods.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T114298>.

This example illustrates how to use the PDF Document API component for merging pages of two separate PDF files into a single PDF file.

C#

(Program.cs)
using DevExpress.Pdf;
namespace PdfMergeExample {
 class Program {
 static void Main(string[] args) {
 using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) {
 pdfDocumentProcessor.CreateEmptyDocument("..\\..\\docs\\Merged.pdf");
 pdfDocumentProcessor.AppendDocument("..\\..\\docs\\TextMerge1.pdf");
 pdfDocumentProcessor.AppendDocument("..\\..\\docs\\TextMerge2.pdf");
 }
 }
 }
}

Visual Basic

(Program.vb)
Imports DevExpress.Pdf
Namespace PdfMergeExample
 Friend Class Program
 Shared Sub Main(ByVal args() As String)
 Using pdfDocumentProcessor As New PdfDocumentProcessor()
 pdfDocumentProcessor.CreateEmptyDocument("..\\..\\docs\\Merge
 pdfDocumentProcessor.AppendDocument("..\\..\\docs\\TextMerge1
 pdfDocumentProcessor.AppendDocument("..\\..\\docs\\TextMerge2
 End Using
 End Sub
 End Class
End Namespace

Page Manipulation

[Office File API](#) > [PDF Document API](#) > [Document Manipulation](#) > [Page Manipulation](#)

The topic describes the following page manipulation capabilities:

- [Copying pages from one document to another PDF](#)
- [Adding/Inserting a new page into a document](#)
- [Rendering a new page with graphics](#)
- [Deleting a page from a document](#)
- [Rotating a page in a document](#)

Copying pages from one document to another PDF

To copy pages:

- Create two [PdfDocumentProcessor](#) instances;
- Load a source document to the first instance and a target document to the second by calling the [PdfDocumentProcessor.LoadDocument](#) overload method;
- Retrieve pages within the source document using the PdfDocument.Pages property.

This property returns an IList<PdfPage> object. You can use the collection methods to perform most page manipulations (for example, split one document into multiple PDF files or add an extracted page to another PDF). To access the document, use the [PdfDocumentProcessor.Document](#) property.

- Save the resulting document by calling the [PdfDocumentProcessor.SaveDocument](#) overload method.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T581087>.

This example shows how to copy a page from one PDF document to another PDF.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace CopyPage {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor source = new PdfDocumentProcessor()) {
                source.LoadDocument("..\\..\\..\\Document1.pdf");
                using (PdfDocumentProcessor target = new PdfDocumentProcessor()) {
                    target.LoadDocument("..\\..\\..\\Document2.pdf");
                    target.Document.Pages.Insert(3, source.Document.Pages[0]);
                    target.SaveDocument("..\\..\\..\\Result.pdf");
                }
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace CopyPage
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using source As New PdfDocumentProcessor()
                source.LoadDocument("..\\..\\Document1.pdf")
                Using target As New PdfDocumentProcessor()
                    target.LoadDocument("..\\..\\Document2.pdf")
                    target.Document.Pages.Insert(3, source.Document.Pages(0))
                    target.SaveDocument("..\\..\\Result.pdf")
                End Using
            End Using
        End Sub
    End Class
End Namespace
```

Adding/Inserting a new page into a document

To add a new page to the end of a document, call the [PdfDocumentProcessor.AddNewPage](#) method with the specified page size, as shown below.

To insert a new page into a document, call the [PdfDocumentProcessor.InsertNewPage](#) method with the page number where the new page should be inserted and the page size.

To save the resulting document, call the [PdfDocumentProcessor.SaveDocument](#) overload method.

The following code shows how to add a new page to a document:

C#

```
using DevExpress.Pdf;
namespace PdfExample {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcess
                processor.LoadDocument("..\\..\\Document.pdf");
                processor.AddNewPage(PdfPaperSize.A4);
                processor.SaveDocument("..\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
Namespace PdfExample
    Class Program
        Private Shared Sub Main(args As String())
            Using processor As New PdfDocumentProcessor()
                processor.LoadDocument("../..\Document.pdf")
                processor.AddNewPage(PdfPaperSize.A4)
                processor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

The graphics content (for example, an image or string) can be added to an existing page's foreground/background by calling the PdfGraphics.AddToPageForeground or PdfGraphics.AddToPageBackground overload method.

The PDF graphics is represented by an instance of the PdfGraphics class. To access this class, you need to reference the **DevExpress.Pdf.Drawing** assembly. See the [Adding Graphics Content to a Page](#) topic to learn more.

Rendering a new page with graphics

The [PdfDocumentProcessor](#) class contains the [PdfDocumentProcessor.RenderNewPage](#) overload methods that render a new page with graphics content.

See the [PDF Graphics](#) section to learn more about PDF Graphics.

Deleting a page from a document

To delete a particular page from a document, call the [PdfDocumentProcessor.DeletePage](#) method with the number of the page that should be deleted.

Example

This example shows how to delete odd-numbered pages in the document, starting from the last odd-numbered page.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T114310>.

C#

```
using DevExpress.Pdf;
// ...
class Program {
    static void Main(string[] args) {
        using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocu
            pdfDocumentProcessor.LoadDocument("..\\..\\TextDelete.pdf")
            for (int i = pdfDocumentProcessor.Document.Pages.Count / 2
                pdfDocumentProcessor.DeletePage(i * 2 + 1);
            pdfDocumentProcessor.SaveDocument("..\\..\\Deleted.pdf");
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
' ...
Friend Class Program
    Shared Sub Main(ByVal args() As String)
        Using pdfDocumentProcessor As New PdfDocumentProcessor()
            pdfDocumentProcessor.LoadDocument("..\\..\\TextDelete.pdf")
            For i As Integer = pdfDocumentProcessor.Document.Pages.Cou
                pdfDocumentProcessor.DeletePage(i * 2 + 1)
            Next i
            pdfDocumentProcessor.SaveDocument("..\\..\\Deleted.pdf")
        End Using
    End Sub
End Class
```

Rotating a page in a document

To rotate a page at a specified angle:

- Load an original document into the PDF Document API component by calling the overloaded [PdfDocumentProcessor.LoadDocument](#) method;
- Retrieve a page by iterating through the page collection the PdfDocument.Pages property returns. To access the document, use the [PdfDocumentProcessor.Document](#) property;
- Specify the rotation angle (in degrees) using the PdfPageTreeObject.Rotate property.
- Save the resulting document by calling the [PdfDocumentProcessor.SaveDocument](#) overload method.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T114305>.

This example illustrates how to use the PDF Document API component for rotating pages.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace PdfPageRotationExample {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocu
                pdfDocumentProcessor.LoadDocument("../\\..\\docs\\TextRotat
                int angle = 0;
                foreach (PdfPage page in pdfDocumentProcessor.Document.Pag
                    angle = (angle + 90) % 360;
                    page.Rotate = angle;
                }
                pdfDocumentProcessor.SaveDocument("../\\..\\docs\\Rotated.p
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace PdfPageRotationExample
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using pdfDocumentProcessor As New PdfDocumentProcessor()
                pdfDocumentProcessor.LoadDocument("../\\..\\docs\\TextRotate.p
                Dim angle As Integer = 0
                For Each page As PdfPage In pdfDocumentProcessor.Document.
                    angle = (angle + 90) Mod 360
                    page.Rotate = angle
                Next page
                pdfDocumentProcessor.SaveDocument("../\\..\\docs\\Rotated.pdf"
            End Using
        End Sub
    End Class
End Namespace
```

Additional Content

[Office File API](#) > [PDF Document API](#) > [Additional Content](#)

The topics in this section:

- [Bookmarks](#)
Create a bookmark, change bookmark attributes (for example, rename a bookmark, change its destination or font style) and delete bookmarks.
- [Hyperlinks](#)
Add a hyperlink to a page or URI.
- [Attachments](#)
Attach any number of files to the PDF document and access attached files.

Bookmarks

[Office File API](#) > [PDF Document API](#) > [Additional Content](#) > [Bookmarks](#)

The PDF Document API provides properties and methods for generating new bookmarks or editing existing document's bookmarks in code.

Overview

A PdfBookmark class instance represents a bookmark. It can be accessed as an item of the PdfBookmark objects list returned by the PdfDocument.Bookmarks property. The **PdfBookmark** class contains the following properties to customize bookmarks:

Member	Description
PdfBookmark.Action	Provides access to the bookmark action being executed.
PdfBookmark.Children	Gets or sets the collection of bookmark children for a document with a tree-structured hierarchy.
PdfBookmark.Destination	Gets or sets a destination (a particular view of a document) to which a bookmark is referred to.
PdfBookmark.IsBold	Gets or sets the value indicating whether the bookmark text is bold.
PdfBookmark.IsItalic	Gets or sets the value indicating whether the bookmark text is italic.
PdfBookmark.IsInitiallyClosed	Gets or sets a value that indicates whether bookmarks are initially closed (bookmark children are hidden) in the navigation panel after a document is loaded.
PdfBookmark.TextColor	Gets or sets the color for a bookmark's text in the navigation pane.
PdfBookmark.Title	Gets or sets the bookmark's text on the navigation pane.

Creating a Bookmark Destination

Call an overloaded [PdfDocumentProcessor.CreateDestination](#) method to create a destination to which a bookmark should be linked. The PDF Document API component can create following types of the destinations:

- PdfFitBBoxDestination
- PdfFitBBoxHorizontallyDestination
- PdfFitBBoxVerticallyDestination
- PdfFitHorizontallyDestination
- PdfFitRectangleDestination
- PdfFitVerticallyDestination
- PdfXYZDestination

The created destination is assigned to the PdfBookmark.Destination property.

The overloaded [PdfDocumentProcessor.CreateDestination](#) methods use the world coordinate system. The world coordinate system's origin (0,0) is at the top left of the page. A positive x increases towards the right and a positive y towards the bottom. So, if you want to pass the page coordinates for a destination to one of the [PdfDocumentProcessor.CreateDestination](#) methods as arguments, you need to convert these page coordinates to world coordinates. See [Coordinate Systems](#) for more information.

When the overloaded [PdfDocumentProcessor.CreateDestination](#) methods create a destination represented by the PdfDestination object, the world coordinates are converted to page coordinates (see [Coordinate Systems](#) for details). To convert the world coordinates (units) into a physical value such as inches, these methods need the resolution's value (dpi) to be passed as **dpiX** and **dpiY** parameters. The default resolution is **96** dpi if no value is passed to these methods.

Since the measurement unit in the page coordinate system is a point (1/72 of an inch), pass **72** as **dpiX** and **dpiY** values to one of the [PdfDocumentProcessor.CreateDestination](#) methods.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T495077>.

This example shows how to create bookmarks in code and add them to a document.

To do this:

- Create a PdfBookmark object;
- Specify the bookmark title and destination using the PdfBookmark.Title and PdfBookmark.Destination properties. To create a destination, call a PdfDocumentProcessor.CreateDestination overloaded method.
- Add the bookmark to the bookmarks collection, which is accessed from the PdfDocument.Bookmarks property.

Note

The measurement unit in the destination is equal to **1/72** of an inch.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace AddBookmarks {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../\\..\\Document.pdf");
                // Create bookmarks and add them to the PDF document.
                PdfDestination destination1 = processor.CreateDestination(1, 180, 150);
                PdfDestination destination2 = processor.CreateDestination(1, 168, 230);
                PdfDestination destination3 = processor.CreateDestination(1, 20, 350);
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "PDF Document Processor", Destination = destination1 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Display, Print and Export PDF", Destination = destination2 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Learn More", Destination = destination3 });
                // Save the result document.
                processor.SaveDocument("../\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace AddBookmarks
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Create bookmarks and add them to the PDF document.
                Dim destination1 As PdfDestination = processor.CreateDesti
                Dim destination2 As PdfDestination = processor.CreateDesti
                Dim destination3 As PdfDestination = processor.CreateDesti
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.
                ' Save the result document.
                processor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Hyperlinks](#)

[Attachments](#)

[How to: Bookmark Search Results in a Document](#)

Hyperlinks

[Office File API](#) > [PDF Document API](#) > [Additional Content](#) > [Hyperlinks](#)

You can add links to a page and the URI using [PDF Graphics](#) that are represented by an instance of the PdfGraphics class.

Note

To access the PdfGraphics class, you need to reference the **DevExpress.Pdf.Drawing** assembly.

The PdfGraphics class methods (including PdfGraphics.AddLinkToPage and PdfGraphics.AddLinkToUri) use the world coordinate system. The origin (0,0) is at the top left of the page. Positive x increases towards the right and positive y - towards the bottom. See [Coordinate Systems](#) to learn more about coordinate systems.

To add a link to a page, call one of the overloaded PdfGraphics.AddLinkToPage methods.

The link to the URI is created using the URI and link area as arguments of the PdfGraphics.AddLinkToUri method.

The coordinates of a hyperlink rectangle is specified in the **RectangleF** object. The hyperlink rectangle is measured in PDF measurement units (points - 1/72 of an inch).

To render a page with created graphics (see [Adding Graphics Content to a Page](#) for more details), call one of the PdfDocumentProcessor.RenderNewPage overloaded method. This method allows you to optimize memory usage (the resources that belong to different pages are not doubled).

The DPI can be passed as dpiX and dpiY parameters to one of the PdfDocumentProcessor.RenderNewPage overloaded methods. The default resolution is **96** dpi.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T495074>.

This example shows how to add a hyperlink to a URI using the PdfGraphics.AddLinkToUri method.

C#

```
(Program.cs)
using System;
using DevExpress.Pdf;
using System.Drawing;
namespace AddLinkToUri {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw graphics.
                using (PdfGraphics graphics = processor.CreateGraphics()) {
                    DrawGraphics(graphics);
                    // Create a link to URI specifying link area and URI.
                    graphics.AddLinkToUri(new RectangleF(310, 150, 180, 15), new Uri("https://www.devexpress.com"));
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graphics) {
            // Draw a text line on the page.
            using (Font font = new Font("Arial", 10)) {
                SolidBrush blue = (SolidBrush)Brushes.Blue;
                graphics.DrawString("https://www.devexpress.com", font, blue, 310, 150);
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports System
Imports DevExpress.Pdf
Imports System.Drawing
Namespace AddLinkToUri
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw graphics.
                Using graphics As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graphics)
                    ' Create a link to URI specifying link area and URI.
                    graphics.AddLinkToUri(New RectangleF(310, 150, 180, 15
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graphics As PdfGraphics)
            ' Draw a text line on the page.
            Using font As New Font("Arial", 10)
                Dim blue As SolidBrush = CType(Brushes.Blue, SolidBrush)
                graphics.DrawString("https://www.devexpress.com", font, bl
            End Using
        End Sub
    End Class
End Namespace
```

See Also[Attachments](#)[Bookmarks](#)[How to: Add a Link to a Page](#)

Attachments

[Office File API](#) > [PDF Document API](#) > [Additional Content](#) > [Attachments](#)

File attachments are represented by an instance of the PdfFileAttachment class. You can specify the attachment creation date, description, and file name using PdfFileAttachment.CreationDate, PdfFileAttachment.Description, and PdfFileAttachment.FileName properties. You also need to specify the file data for the attachment using the PdfFileAttachment.Data property.

if required, you can specify additional properties of a file attachment such as a file's MIME type and relationship using PdfFileAttachment.MimeType, and PdfFileAttachment.Relationship properties.

To attach a file to a document, call the PdfDocumentProcessor.AttachFile method.

To get an attachment, use the PdfDocument.FileAttachments property.

To delete an attachment, call the PdfDocumentProcessor.DeleteAttachment method.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T304234>.

This example shows how to programmatically attach a file to the PDF document.

To do this:

- Create a PdfFileAttachment object;
- Specify the attachment creation date, description, and file name using PdfFileAttachment.CreationDate, PdfFileAttachment.Description, and PdfFileAttachment.FileName properties. To specify the data for the attachment, use the PdfFileAttachment.Data property;
- If required, you can specify additional properties of an attached file, for example, the file's mime type and relationship using PdfFileAttachment.MimeType and PdfFileAttachment.Relationship properties, respectively.
- Call the PdfDocumentProcessor.AttachFile method with a file attachment object used as a parameter.
- Save the attachment to a document by calling the PdfDocumentProcessor.SaveDocument method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.IO;
namespace AttachFile {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../\\..\\Document.pdf");
                // Attach a file to the PDF document.
                processor.AttachFile(new PdfFileAttachment() {
                    CreationDate = DateTime.Now,
                    Description = "This is my attach file.",
                    FileName = "MyAttach.txt",
                    Data = File.ReadAllBytes("../\\..\\FileToAttach.txt")
                });
                // The attached document.
                processor.SaveDocument("../\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.IO
Namespace AttachFile
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Attach a file to the PDF document.
                processor.AttachFile(New PdfFileAttachment() With { _
                    .CreationDate = Date.Now, _
                    .Description = "This is my attach file.", _
                    .FileName = "MyAttach.txt", _
                    .Data = File.ReadAllBytes("../..\FileToAttach.txt") _
                })
                ' The attached document.
                processor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also[Hyperlinks](#)[Bookmarks](#)

Interactive Forms

[Office File API](#) > [PDF Document API](#) > [Interactive Forms](#)

The topics in this section contain all the required information related to an interactive form in a PDF document.

- [Interactive Form Filling](#)
Fill existing interactive forms.
- [Interactive Form Flattening](#)
Flatten interactive forms in a document.
- [Creating Interactive Form](#)
Create interactive form fields and add them to a document.
- [Deleting Interactive Form](#)
Remove interactive form fields from a document.
- [Export and Import Interactive Form Data](#)
Export and import interactive form data in various formats - FDF, XFDF, XML and TXT.

Interactive Form Filling

[Office File API](#) > [PDF Document API](#) > [Interactive Forms](#) > [Interactive Form Filling](#)

This topic describes different ways of interactive form filling.

This document consists of the following sections:

- [Filling of an Interactive Form using Form Field Names](#)
- [Filling of an Interactive Form using Data File](#)

Filling of an Interactive Form using Form Field Names

To load a PDF document that contains an interactive form, create an instance of the PdfDocumentProcessor object and call one of the PdfDocumentProcessor.LoadDocument overloaded methods.

Calling the PdfDocumentProcessor.GetFormData method acquires the interactive form data from a PDF document. This method returns a PdfFormData object that represents interactive form data.

Note

Do not use a previously obtained PdfFormData instance after the interactive form structure is changed. For example, after one of the following operations is applied: adding new interactive form fields, interactive form flattening, merging documents with interactive forms, or deleting pages with interactive form fields. You can obtain the new PdfFormData instance by calling the PdfDocumentProcessor.GetFormData method.

To set or update interactive form field values, provide the field name as a key (e.g., data[☐ ☐ `FieldName1`].Value = SomeValue1).

To specify a value for a form field, use the PdfFormData.Value property.

Important

The value specified for an interactive form field can be a string for text fields, a name of the checked/unchecked appearance for radio buttons and check boxes, an array of strings (for multi select fields) or a PdfFormData object (for the complex fields for which the field name is provided as a key).

When all values have been set, call the PdfDocumentProcessor.ApplyFormData method to apply form data to the interactive form.

Call one of the PdfDocumentProcessor.SaveDocument overloaded methods to commit these changes to a PDF document.

See the code snippet below.

C#

```
using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
    processor.LoadDocument(pathForPdf);
    PdfFormData data = processor.GetFormData();
    data["Your_Field_Name_1"].Value = "Some Value 1";
    data["Your_Field_Name_2"].Value = "Some Value 2";
    data["Your_Field_Name_3"].Value = "Some Value 3";
    processor.ApplyFormData(data);
    processor.SaveDocument(pathForPdf);
}
```

Visual Basic

```
Using processor As New PdfDocumentProcessor()
    processor.LoadDocument(pathForPdf)
    Dim data As PdfFormData = processor.GetFormData()
    data("Your_Field_Name_1").Value = "Some Value 1"
    data("Your_Field_Name_2").Value = "Some Value 2"
    data("Your_Field_Name_3").Value = "Some Value 3"
    processor.ApplyFormData(data)
    processor.SaveDocument(pathForPdf)
End Using
```

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

If required, you can easily retrieve a list of field name by iterating through the string collection returned by the [PdfDocumentProcessor.GetFormFieldNames](#) method.

```
C#
foreach (string name in formData.GetFieldNames())
{
    object value = formData[name].Value;
}
```

Visual Basic

```
For Each name As String In formData.GetFieldNames()
    Dim value As Object = formData(name).Value
Next
```

Filling of an Interactive Form using Data File

If you have a file with interactive form data (Fdf, Xml, Xfdf or Txt), you can fill the PdfFormData object directly from this file. To do this, pass the name of the data file and format when creating a **PdfFormData** object.

Note

If you omit the data format parameter, the PDF Document API component automatically detects this type.

See the code snippet below.

```
C#
using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
    processor.LoadDocument(pathForPdf);
    PdfFormData data = new PdfFormData(pathForData); //automatic detection of type
    processor.ApplyFormData(data);
    processor.SaveDocument(pathForPdf);
}
```

Visual Basic

```
Using processor As New PdfDocumentProcessor()
    processor.LoadDocument(pathForPdf)
    Dim data As New PdfFormData(pathForData)
    'automatic detection of type
    processor.ApplyFormData(data)
    processor.SaveDocument(pathForPdf)
End Using
```

See Also

[How to: Obtain a Checked Appearance Name for a Check Box](#)
[Interactive Form Flattening](#)
[Creating Interactive Form](#)
[Deleting Interactive Form](#)
[Export and Import Interactive Form Data](#)

Interactive Form Flattening

[Office File API](#) > [PDF Document API](#) > [Interactive Forms](#) > [Interactive Form Flattening](#)

This document explains interactive form flattening and provides information on how to flatten interactive forms using the PDF Document API component.

This document consists of the following sections.

- [Overview](#)
- [Flattening an Entire Form](#)
- [Flattening a Specific Field by its Name](#)

Overview

A PDF document can contain interactive forms (AcroForms) with form fields (e.g., text fields, buttons, list boxes). You can edit an interactive form (e.g., click on the form field and change its value, or fill it out).

The screenshot shows an 'Arrival Card' form with the following fields:

- Last Name:** Text field containing 'Leverling'.
- First Name:** Text field containing 'Janet'.
- Nationality:** Dropdown menu with 'United States' selected. The dropdown list is open, showing options: Switzerland, Taiwan, Tajikistan, Thailand, Turkey, United Kingdom, and United States.
- Passport No.:** Text field containing '31195855'.
- Visa No.:** Text field containing '73203393'.
- Gender:** Two checkboxes. 'Male' is unchecked, and 'Female' is checked with a checkmark.
- Date of Birth:** Three text fields containing '30', '08', and '1985' respectively.
- Address:** Text field (empty).
- Flight No.:** Text field (empty).

For more information about filling a form with data, see the [How to: Fill an Interactive Form with Values](#) example.

The PDF Document API component can flatten these form fields. The flattening process removes the form field interactive features, so the value entered or selected in the form field will be rendered like regular content (e.g., text, images, shapes, and so on).

Arrival Card

Last Name	<input type="text" value="Leverling"/>	Passport No.	<input type="text" value="31195855"/>
First Name	<input type="text" value="Janet"/>	Visa No.	<input type="text" value="73203393"/>
Nationality	<input type="text" value="United States"/>	Gender	<input type="checkbox"/> Male <input checked="" type="checkbox"/> Female
Address	<input type="text" value="98033, 722 Moss Bay Blvd., Kirkland, WA, USA"/>	Date of Birth	<input type="text" value="30"/> <input type="text" value="08"/> <input type="text" value="1985"/>
Flight No.	<input type="text" value="(KL) KLM 876"/>		

Note

The flattening process cannot be **undone**, so it is recommended that you save the flattened document under a different name.

You can flatten interactive forms in two ways.

- Flattening an entire form
- Flattening a form field by its name

The sections below detail each approach to flattening.

Flattening an Entire Form

To flatten an entire form, call the [PdfDocumentProcessor.FlattenForm](#) method. This method returns **false** if the document does not contain an interactive form to be flattened, otherwise it returns **true**.

See the code snippet below.

C#

```
using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {  
    processor.LoadDocument("FormDemo.pdf");  
    if (processor.FlattenForm())  
        processor.SaveDocument("Result.pdf");  
}
```

Visual Basic

```
Using processor As New PdfDocumentProcessor()  
    processor.LoadDocument("FormDemo.pdf")  
    If processor.FlattenForm() Then  
        processor.SaveDocument("Result.pdf")  
    End If  
End Using
```

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T446039>.

Flattening a Specific Field by its Name

If you need to flatten specific fields, you can call the [PdfDocumentProcessor.FlattenFormField](#) method and pass the name of the field to it. This method returns **false**, if the form field was not found in the document; otherwise it returns - **true**.

Note that the [PdfDocumentProcessor.FlattenFormField](#) method flattens only a field whose name is known. Other form fields maintain their interactive features.

To obtain a list of form fields names, use the [PdfDocumentProcessor.GetFormFieldNames](#) method.

Creating Interactive Form

[Office File API](#) > [PDF Document API](#) > [Interactive Forms](#) > [Creating Interactive Form](#)

This topic describes how to create interactive form fields and add them to an existing PDF document.

Note

To learn how to create interactive form fields, see the [Adding Interactive Form Fields](#) topic.

Before adding interactive form fields to a PDF document, create an instance of the [PdfDocumentProcessor](#) class and load a PDF document into the PDF Document API component using one of the overloaded [PdfDocumentProcessor.LoadDocument](#) methods.

An instance of the PdfAcroFormField object represents the interactive form field.

To create an interactive form field, do one of the following:

- create an instance of the class that represents the specific form field (for example, PdfAcroFormTextBoxField);
- call the corresponding static **Create** method for the PdfAcroFormField class (for example, PdfAcroFormField.CreateTextBox).

The PdfAcroFormField.CreateTextBox method returns a PdfAcroFormTextBoxField object that represents a created text box field. This method needs the name of the field, a PdfRectangle instance that defines the position of the form field on a document page, and a page number.

Then, specify the interactive form field properties. For example, specify the text box text, and type using PdfAcroFormTextBoxField.Text, and PdfAcroFormTextBoxField.Type properties, respectively.

To specify the text box name, tooltip and appearance settings, use the PdfAcroFormField.Name, PdfAcroFormField.ToolTip, and PdfAcroFormVisualField.Appearance properties.

Note

Form field names must be unique in an interactive form.

To find a collision in the interactive form field names, call one of the overloaded [PdfDocumentProcessor.CheckFormFieldNameCollisions](#) methods and pass interactive form fields that should be checked as a parameter. These methods return a collection of PdfAcroFormFieldNameCollision objects that contains information about a collision found in interactive form field names.

To obtain the interactive form field in which a collision was found with a field name, use the PdfAcroFormFieldNameCollision.Field property.

You can also get the forbidden field names using the PdfAcroFormFieldNameCollision.ForbiddenNames property.

To add interactive form fields to a document, pass a collection of form fields (for example, PdfAcroFormTextBoxField objects) as a parameter to one of the overloaded [PdfDocumentProcessor.AddFormFields](#) methods.

To save a document, call one of the overloaded [PdfDocumentProcessor.SaveDocument](#) methods.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494642>.

This example shows how to create interactive form fields (e.g., text box and radio button group fields) and add them to a document.

C#

```

(Program.cs)
using DevExpress.Pdf;
namespace AddFormFieldsToExistingDocument {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../\\..\\Document.pdf");
                // Create a text box field specifying the field name, page number, and field location on the page.
                PdfAcroFormTextBoxField textBox = new PdfAcroFormTextBoxField("text box", 1, new PdfRectangle(230, 635, 250, 655));
                // Specify text box text, and appearance.
                textBox.Text = "Text Box";
                textBox.Appearance.BackgroundColor = new PdfRGBColor(0.8, 0.5, 0.3);
                textBox.Appearance.FontSize = 12;
                // Create a radio group field specifying its name and the page number.
                PdfAcroFormRadioGroupField radioGroup = new PdfAcroFormRadioGroupField("Gender Group", 1);
                // Add the first radio button to the group and specify its location using a PdfRectangle object.
                radioGroup.AddButton("button1", new PdfRectangle(230, 635, 250, 655));
                // Add the second radio button to the group.
                radioGroup.AddButton("button2", new PdfRectangle(310, 635, 330, 655));
                // Specify radio group selected index, and appearance.
                radioGroup.SelectedIndex = 0;
                radioGroup.Appearance.BorderAppearance = new PdfAcroFormBorderAppearance() { Color = new PdfRGBColor(0.8, 0.5, 0.3) };
                // Add form fields to the page.
                processor.AddFormFields(textBox, radioGroup);
                // Save the result document.
                processor.SaveDocument("../\\..\\Result.pdf");
            }
        }
    }
}

```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace AddFormFieldsToExistingDocument
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Create a text box field specifying the field name, page
                Dim textBox As New PdfAcroFormTextBoxField("text box", 1,
                ' Specify text box text, and appearance.
                textBox.Text = "Text Box"
                textBox.Appearance.BackgroundColor = New PdfRGBColor(0.8,
                textBox.Appearance.FontSize = 12
                ' Create a radio group field specifying its name and the p
                Dim radioGroup As New PdfAcroFormRadioGroupField("Gender G
                ' Add the first radio button to the group and specify its
                radioGroup.AddButton("button1", New PdfRectangle(230, 635,
                ' Add the second radio button to the group.
                radioGroup.AddButton("button2", New PdfRectangle(310, 635,
                ' Specify radio group selected index, and appearance.
                radioGroup.SelectedIndex = 0
                radioGroup.Appearance.BorderAppearance = New PdfAcroFormBo
                ' Add form fields to the page.
                processor.AddFormFields(textBox, radioGroup)
                ' Save the result document.
                processor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Deleting Interactive Form](#)

[Interactive Form Flattening](#)

[Export and Import Interactive Form Data](#)

Deleting Interactive Form

[Office File API](#) > [PDF Document API](#) > [Interactive Forms](#) > [Deleting Interactive Form](#)

The PDF Document API provides methods that allow you to remove an interactive form field or a whole interactive form from a PDF document.

Removing Form Field

To remove a particular interactive form field by its name:

- Create an instance of the [PdfDocumentProcessor](#) class;
- Load a PDF document that contains an interactive form field using one of the overloaded [PdfDocumentProcessor.LoadDocument](#) methods;
- If you do not know the name of an interactive form field that should be removed, obtain names of interactive form fields by calling the [PdfDocumentProcessor.GetFormFieldNames](#) method;
- Remove a desired form field in the interactive form. To do this, call the [PdfDocumentProcessor.RemoveFormField](#) method and pass a field name as an argument to this method.

This method tries to find the field in a document using the field name. If it finds this field, the method removes it from the document and returns **true**; otherwise, it returns **false**.

See the code snippet below.

```
C#

using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
    // Load a document with an interactive form.
    processor.LoadDocument("..\\..\\InteractiveForm.pdf");
    // Remove a form field by its name.
    if (processor.RemoveFormField("FirstName"))
        // Save the modified document.
        processor.SaveDocument("..\\..\\Result.pdf");
    else
        // Shows a message if the form field is not found in a document.
        Console.WriteLine("The form field was not removed from a document. Make sure, the form field exist");
}
```

Visual Basic

```
Using processor As New PdfDocumentProcessor()
    ' Load a document with an interactive form.
    processor.LoadDocument("..\\..\\InteractiveForm.pdf")
    ' Remove a form field by its name.
    If processor.RemoveFormField("FirstName") Then
        ' Save the modified document.
        processor.SaveDocument("..\\..\\Result.pdf")
    Else
        ' Show a message if the form field was not found in a document.
        Console.WriteLine("The form field was not removed from a document. Make sure, the form field exist")
    End If
End Using
```

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494240>.

Removing Interactive Form

To remove all form fields from a document:

- Create an instance of the [PdfDocumentProcessor](#) class;
- Load a PDF document that contains an interactive form field using one of the overloaded [PdfDocumentProcessor.LoadDocument](#) methods;
- Call the [PdfDocumentProcessor.RemoveForm](#) method.

If the interactive form is successfully removed, this method returns **true**. If the document does not contain an interactive form that should be removed, it returns **false**.

See the code snippet below.

```
C#
using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
    // Load a document with an interactive form.
    processor.LoadDocument("../..\\InteractiveForm.pdf");
    // Remove the whole interactive form.
    if (processor.RemoveForm())
        // Save the modified document.
        processor.SaveDocument("../..\\Result.pdf");
    else
        // Shows a message if the interactive form is not found in a document.
        Console.WriteLine("The interactive form was not removed from a document. Make sure the interactive
}
}
```

Visual Basic

```
Using processor As New PdfDocumentProcessor()
    ' Load a document with an interactive form.
    processor.LoadDocument("../..\\InteractiveForm.pdf")
    ' Remove the whole interactive form.
    If processor.RemoveForm() Then
        ' Save the modified document.
        processor.SaveDocument("../..\\Result.pdf")
    Else
        ' Show a message if the interactive form was not found in a document.
        Console.WriteLine("The interactive form was not removed from a document. Make sure the interactive
    End If
End Using
```

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494240>.

Export and Import Interactive Form Data

[Office File API](#) > [PDF Document API](#) > [Interactive Forms](#) > [Export and Import Interactive Form Data](#)

This topic describes how to export and import AcroForm data (interactive form data) in various formats.

Export Interactive Form Data

Before exporting interactive form data, make sure the interactive form contains fields with filled values. To learn how to fill an interactive form with values, see the [How to: Fill an Interactive Form with Values](#) example.

To load a document with the required interactive form, call the [PdfDocumentProcessor.LoadDocument](#) method.

To export interactive form data from a document to FDF, XFDF, XML or TXT format, call one of the overloaded [PdfDocumentProcessor.Export](#) methods with the specified data format settings.

The following example shows how to export interactive form data from a PDF document to XML format.

C#

```
using DevExpress.Pdf;
namespace ExportInteractiveForms {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a PDF document with AcroForm data.
                processor.LoadDocument("../..\..\InteractiveForm.pdf");
                // Export AcroForm data to XML format.
                processor.Export("../..\..\InteractiveForm.xml", PdfFormDateFormat.Xml);
            }
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
Namespace ExportInteractiveForms
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a PDF document with AcroForm data.
                processor.LoadDocument("../..\..\InteractiveForm.pdf")
                ' Export AcroForm data to XML format.
                processor.Export("../..\..\InteractiveForm.xml", PdfFormDateFormat.Xml)
            End Using
        End Sub
    End Class
End Namespace
```

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T491302>.

You can also save the form data to the specified path and format calling one of the overloaded PdfFormData.Save methods.

To obtain interactive form data, call the [PdfDocumentProcessor.GetFormData](#) method.

See the code snippet below.

C#

```
using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
    processor.LoadDocument(pathForPdf);
    PdfFormData data = processor.GetFormData();
    data.Save(pathForExportedData, PdfFormDateFormat.Fdf);
}
```

Visual Basic

```

Using processor As New PdfDocumentProcessor()
    processor.LoadDocument(pathForPdf)
    Dim data As PdfFormData = processor.GetFormData()
    data.Save(pathForExportedData, PdfFormDataFormat.Fdf)
End Using

```

Import Interactive Form Data

To load a document with an interactive form, call the [PdfDocumentProcessor.LoadDocument](#) method.

To import interactive form data either from FDF, XFDF, XML or TXT format to a PDF document, call one of the overloaded [PdfDocumentProcessor.Import](#) methods with data format settings or without them.

Note

The previous interactive form data is automatically removed from a document after the import operation is performed.

To save the document, call one of the overloaded [PdfDocumentProcessor.SaveDocument](#) methods.

The example below demonstrates how to import interactive form data from XML format.

C#

```

using DevExpress.Pdf;
namespace ImportInteractiveForms {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a PDF document with AcroForm data.
                processor.LoadDocument("../..\..\EmptyForm.pdf");
                // Import AcroForm data from an XML file.
                processor.Import("../..\..\InteractiveForm.xml");
                // Save the imported document.
                processor.SaveDocument("../..\..\InteractiveForm.pdf");
            }
        }
    }
}

```

Visual Basic

```

Imports DevExpress.Pdf
Namespace ImportInteractiveForms
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a PDF document with AcroForm data.
                processor.LoadDocument("../..\..\EmptyForm.pdf")
                ' Import AcroForm data from an XML file.
                processor.Import("../..\..\InteractiveForm.xml")
                ' Save the imported document.
                processor.SaveDocument("../..\..\InteractiveForm.pdf")
            End Using
        End Sub
    End Class
End Namespace

```

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T491383>.

Text Markup Annotations

[Office File API](#) > [PDF Document API](#) > [Text Markup Annotations](#)

The PDF Document API component provides an API that can be used to manipulate (create, modify or remove) text markup annotations in pages.

You can apply the following markup annotation types to a text in a document: underline, highlight, strikeout, and squiggly underline.

The topics in this section contain all the required information related to text markup annotation manipulation in a PDF document.

- [Creating](#)
Create markup annotations for text and add them to a page.
- [Modifying](#)
Change existing text markup annotations on a page.
- [Deleting](#)
Remove text markup annotations from a document page.

Creating

[Office File API](#) > [PDF Document API](#) > [Text Markup Annotations](#) > [Creating](#)

You can add the following text markup annotations to a page: highlight, strikethrough, underline, and squiggly underline.

To add a text markup annotation to a page, call one of the overloaded [PdfDocumentProcessor.AddTextMarkupAnnotation](#) methods, where specify the annotation type, page number and a page area corresponding to the text that should be annotated on this page. These methods return the PdfTextMarkupAnnotationData object that represents the markup annotation data.

Note

If a specified page area does not correspond to text on the page, the annotation is not created and overloaded [PdfDocumentProcessor.AddTextMarkupAnnotation](#) methods return **null**.

Then, you can specify the annotation settings using the PdfMarkupAnnotationData class's properties.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T545395>.

This example shows how to create a text markup annotation that highlights text in a document and specify the annotation properties.

C#

```
using System;
using DevExpress.Pdf;
namespace CreateMarkupAnnotation {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../\\..\\Document.pdf");
                // Add a text markup annotation to the first page for text corresponding to the specified
                PdfTextMarkupAnnotationData annot = processor.AddTextMarkupAnnotation(1, new PdfRectangle
                    PdfTextMarkupAnnotationType.Highlight);
                if (annot != null) {
                    // Specify the annotation properties.
                    annot.Author = "Bill Smith";
                    annot.Contents = "Note";
                    annot.Color = new PdfRGBColor(0.8, 0.2, 0.1);
                    // Save the resulting document.
                    processor.SaveDocument("../\\..\\Result.pdf");
                }
                else
                    Console.WriteLine("The annotation cannot be added to a page. Make sure the specified p
            }
        }
    }
}
```

Visual Basic

```
Imports System
Imports DevExpress.Pdf
Namespace CreateMarkupAnnotation
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Add a text markup annotation to the first page for text corresponding to the specified p
                Dim annot As PdfTextMarkupAnnotationData = processor.AddTextMarkupAnnotation(1, New PdfRec
                If annot IsNot Nothing Then
                    ' Specify the annotation properties.
                    annot.Author = "Bill Smith"
                    annot.Contents = "Note".
                    annot.Color = New PdfRGBColor(0.8, 0.2, 0.1)
                    ' Save the resulting document.
                    processor.SaveDocument("../..\Result.pdf")
                Else
                    Console.WriteLine("The annotation cannot be added to a page. Make sure the specified p
                End If
            End Using
        End Sub
    End Class
End Namespace
```

See Also

PdfMarkupAnnotationData

Modifying

[Office File API](#) > [PDF Document API](#) > [Text Markup Annotations](#) > [Modifying](#)

A collection of PdfMarkupAnnotationData objects represents the text markup annotations' data. To obtain this collection for a specific page, call the PdfDocumentProcessor.GetMarkupAnnotationData method and pass the page's number as its parameter. Then, you can change the annotation settings using the PdfMarkupAnnotationData class's properties.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T545429>.

This example shows how to change an existing markup annotation's settings in a PDF document. To retrieve all text markup annotations in a page, call the PdfDocumentProcessor.GetMarkupAnnotationData method.

C#

(Program.cs)
using DevExpress.Pdf;
using System;
using System.Collections.Generic;
namespace ModifyExistingMarkupAnnotation {
 class Program {
 static void Main(string[] args) {
 // Create a PDF Document Processor.
 using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
 // Load a document.
 processor.LoadDocument("../\\..\\Document.pdf");
 // Get a list of annotations in the first document page.
 IList<PdfMarkupAnnotationData> annotations = processor.GetMarkupAnnotationData(1);
 if (annotations.Count > 0) {
 // Change the properties of the first annotation.
 annotations[0].Author = "Bill Smith";
 annotations[0].Contents = "Important!";
 annotations[0].Color = new PdfRGBColor(0.6, 0.7, 0.3);
 annotations[0].Opacity = 0.2;
 // Save the result document.
 processor.SaveDocument("../\\..\\Result.pdf");
 }
 else
 Console.WriteLine("The specified document page does not contain markup annotations!");
 }
 }
 }
}

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Collections.Generic
Namespace ModifyExistingMarkupAnnotation
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Get a list of annotations in the first document page.
                Dim annotations As IList(Of PdfMarkupAnnotationData) = processor.GetAnnotations(0)
                If annotations.Count > 0 Then
                    ' Change the properties of the first annotation.
                    annotations(0).Author = "Bill Smith"
                    annotations(0).Contents = "Important!"
                    annotations(0).Color = New PdfRGBColor(0.6, 0.7, 0.3)
                    annotations(0).Opacity = 0.2
                    ' Save the result document.
                    processor.SaveDocument("../..\Result.pdf")
                Else
                    Console.WriteLine("The specified document page does not contain any annotations.")
                End If
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfMarkupAnnotationData](#)

Deleting

[Office File API](#) > [PDF Document API](#) > [Text Markup Annotations](#) > [Deleting](#)

The PDF Document API component can delete text markup annotations specified in the PdfMarkupAnnotationData collection by calling the PdfDocumentProcessor.DeleteMarkupAnnotations method.

To delete a specific markup annotation from a page, call the PdfDocumentProcessor.DeleteMarkupAnnotation method, passing the PdfMarkupAnnotationData object with annotation data as an argument to this method.

To retrieve the markup annotations, call the PdfDocumentProcessor.GetMarkupAnnotationData method, specifying the page number.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T549297>.

This example shows how to delete text markup annotations created by a particular author.

C#

(Program.cs)
using DevExpress.Pdf;
using System.Linq;
namespace RemoveSpecificMarkupAnnotations {
 class Program {
 static void Main(string[] args) {
 // Create a PDF Document Processor.
 using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
 // Load a document.
 processor.LoadDocument("..\\..\\..\\Document.pdf");
 for (int i = 0; i <= processor.Document.Pages.Count; i++) {
 // Remove Bill Smith's markup annotations from a page.
 processor.DeleteMarkupAnnotations(processor.GetMarkupAnnotationData(i)
 .Where(annotation => annotation.Author.Contains("Bill Smith")));
 // Save the result document.
 processor.SaveDocument("..\\..\\..\\Result.pdf");
 }
 }
 }
 }
}

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Linq
Namespace RemoveSpecificMarkupAnnotations
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                For i As Integer = 0 To processor.Document.Pages.Count
                    ' Remove Bill Smith's markup annotations from a page.
                    processor.DeleteMarkupAnnotations(processor.GetMarkupA
                    ' Save the result document.
                    processor.SaveDocument("../..\Result.pdf")
                Next i
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Creating](#)
[Modifying](#)

Document Security

[Office File API](#) > [PDF Document API](#) > [Document Security](#)

The topics in this section describe how to use PDF save options to protect a PDF document with a password and sign the document with a digital signature.

- [Protecting a Document](#)
Protect a PDF document using both the owner and user passwords. In addition, you can optionally specify user permissions for printing, data extraction, document modification and interactive operations.
- [Signing a Document](#)
Apply a digital signature to a PDF document.

Protecting a Document

[Office File API](#) > [PDF Document API](#) > [Document Security](#) > [Protecting a Document](#)

By default, most documents do not impose restrictions on opening a file used for reading. The **PDF Document API** component can protect a document with a password using both user and owner passwords. Passwords are used to prevent users from accessing or modifying PDF documents.

The sections below provide more detailed descriptions on the user and owner passwords and explain how to customize PDF Document API encryption options to protect a document.

Encryption Settings

Before encrypting, the **PDF Document API** component must have a document (the document can be created by any operation using, for example, the [Document Creation API](#)).

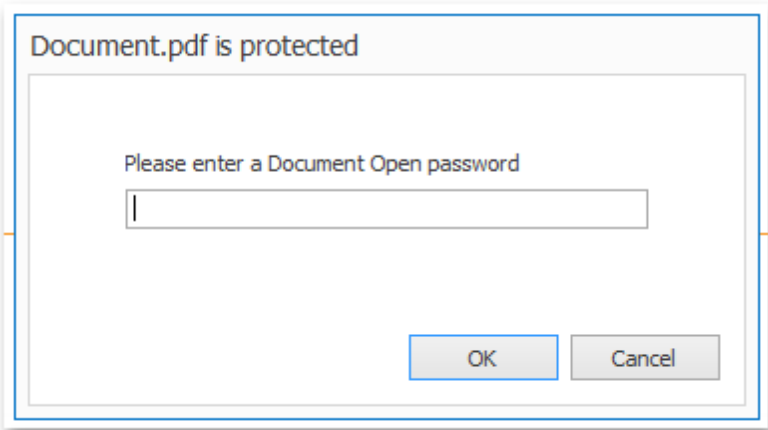
All required settings to protect a document are contained in the PdfEncryptionOptions object, which you can access using the PdfSaveOptions.EncryptionOptions property.

To encrypt the document with these settings, call the overloaded [PdfDocumentProcessor.CreateEmptyDocument](#) or [PdfDocumentProcessor.SaveDocument](#) method and pass the PdfSaveOptions object containing these settings as a parameter.

User password

The **user** password is used when you want to prevent unauthorized access to a document. This password protects sensitive information. So, if you want to open a protected document, you need to enter the user password.

The image below shows a dialog box displayed if you are trying to open a protected document in the PDF Viewer.



Specify the user password using the PdfEncryptionOptions.UserPasswordString property to protect your document from being opened.

Owner password

If a document is opened with a user password or without a password (if the user password is not specified), you can perform only permitted operations. To get full access to a document, open it with the **owner** password. This password can be specified using the PdfEncryptionOptions.OwnerPasswordString property.

Important

If the **Owner** and **User** passwords are the same or the **Owner** password is missing and the document is protected only with the **User** password, the resulting document will have a document opening restriction. In this case, any user will have full access to this document after opening it.

Note

You can control the document access either by restricting or allowing printing, data extraction, document modification and interactive operations.

The restrictions on data extraction, data modification, interactive, or printing operations with a document can't be applied without the owner password.

How to Restrict Operations

To restrict data extraction, data modification, interactive, or printing operations with a PDF document, create a PdfEncryptionOptions object and specify the owner password using the PdfEncryptionOptions.OwnerPasswordString property and permissions listed below.

To encrypt a document with printing permissions, use the PdfEncryptionOptions.PrintingPermissions property. This property can be set to one of the following values.

Name	Description
Allowed	Permit document printing.
LowQuality	Prohibit document printing at the highest quality level.
NotAllowed	Prohibit document printing.

To encrypt a document with data extraction permissions, use the PdfEncryptionOptions.DataExtractionPermissions property. These permissions impose the following restrictions on data extraction operations.

Name	Description
Allowed	Permit data extraction operations (copying, or text and graphics extraction from the document) including access to software that uses assistive technologies.
Accessibility	Allow PDF Viewers to access document content by using the Viewer's accessibility features.
NotAllowed	Prohibit data extraction operations (copying, or text and graphics extraction from the document) including access to software that uses assistive technologies.

To encrypt a document with document modification restrictions, set the PdfEncryptionOptions.ModificationPermissions property to one of the following values.

Name	Description
Allowed	Permit document modification and assembling.
DocumentAssembling	Allow only document assembling such as inserting, rotating or deleting pages, as well as bookmark creation on the navigation pane.
NotAllowed	Prohibit document modification and assembling.

To set interactivity permissions, use the PdfEncryptionOptions.InteractivityPermissions property. The following values are supported.

Name	Description
Allowed	Permit interactive operations (adding or modifying text annotations, filling in interactive form fields, and creating or modifying interactive form fields) in a PDF document.
FormFillingAndSigning	Prohibit interactive operations in a PDF document except form filling in the existing form fields and document signing.
NotAllowed	Prohibit all interactive operations (adding or modifying text annotations, filling in interactive form fields, and creating or modifying interactive form fields) in a PDF document.

Example

This example shows how to protect a PDF document using both the owner and user passwords.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T243764>.

C#

```
using DevExpress.Pdf;
namespace PDFPasswordProtection {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) {
                // Load a PDF document.
                pdfDocumentProcessor.LoadDocument("..\\..\\Demo.pdf");
                // Specify printing, data extraction, modification, and interactivity permissions.
                PdfEncryptionOptions encryptionOptions = new PdfEncryptionOptions();
                encryptionOptions.PrintingPermissions = PdfDocumentPrintingPermissions.Allowed;
                encryptionOptions.DataExtractionPermissions = PdfDocumentDataExtractionPermissions.NotAllowed;
                encryptionOptions.ModificationPermissions = PdfDocumentModificationPermissions.DocumentAsReadOnly;
                encryptionOptions.InteractivityPermissions = PdfDocumentInteractivityPermissions.Allowed;
                // Specify the owner and user passwords for the document.
                encryptionOptions.OwnerPasswordString = "OwnerPassword";
                encryptionOptions.UserPasswordString = "UserPassword";
                // Specify the 256-bit AES encryption algorithm.
                encryptionOptions.Algorithm = PdfEncryptionAlgorithm.AES256;
                // Save the protected document with encryption settings.
                pdfDocumentProcessor.SaveDocument("..\\..\\ProtectedDocument.pdf", new PdfSaveOptions() {
                    // ...
                })
            }
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
Namespace PDFPasswordProtection
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using pdfDocumentProcessor As New PdfDocumentProcessor()
                ' Load a PDF document.
                pdfDocumentProcessor.LoadDocument("..\\..\\Demo.pdf")
                ' Specify printing, data extraction, modification, and interactivity permissions.
                Dim encryptionOptions As New PdfEncryptionOptions()
                encryptionOptions.PrintingPermissions = PdfDocumentPrintingPermissions.Allowed
                encryptionOptions.DataExtractionPermissions = PdfDocumentDataExtractionPermissions.NotAllowed
                encryptionOptions.ModificationPermissions = PdfDocumentModificationPermissions.DocumentAsReadOnly
                encryptionOptions.InteractivityPermissions = PdfDocumentInteractivityPermissions.Allowed
                ' Specify the owner and user passwords for the document.
                encryptionOptions.OwnerPasswordString = "OwnerPassword"
                encryptionOptions.UserPasswordString = "UserPassword"
                ' Specify the 256-bit AES encryption algorithm.
                encryptionOptions.Algorithm = PdfEncryptionAlgorithm.AES256
                ' Save the protected document with encryption settings.
                pdfDocumentProcessor.SaveDocument("..\\..\\ProtectedDocument.pdf", New PdfSaveOptions() With {
                    ' ...
                })
            End Using
        End Sub
    End Class
End Namespace
```

Signing a Document

[Office File API](#) > [PDF Document API](#) > [Document Security](#) > [Signing a Document](#)

This topic explains why it is necessary to sign a document and how this can be done using the **PDF Document API** component.

Overview

The **PDF Document API** allows you to electronically sign a document. A **digital signature** is used to identify the person who had seen a document and authorized it, and also verify document integrity.

To sign a document, you need a digital certificate. The certificate protects document sensitive information from unauthorized access allowing secure document exchange over the Internet. The certificate is issued by a Certification Authority (CA). The role of the CA is to validate the holder's identity and provide access to a certificate.

To learn how to apply a signature to a document, refer to the **How to Sign** document section.

Prerequisite

Before signing, the **PDF Document API** component must have a document (e.g., you can load a document using the [PdfDocumentProcessor.LoadDocument](#) method).

How to Sign

Access a signing certificate.

You can either get it from a third-party certificate authority (CA) or generate the X.509 certificate with a private and public key pair using, e.g, the [Makecert.exe \(Certificate Creation Tool\)](#) for testing purposes only.

Create a signature and specify the signing information.

An electronic signature is represented by an instance of the PdfSignature object with the certificate. The signature can be accessed using the PdfSaveOptions.Signature property.

To help a recipient to verify the signature provided by the signer, you can specify the signer name or organization name, signing location, reason, and contact information using the PdfSignature.Name, PdfSignature.Location, PdfSignature.Reason and PdfSignature.ContactInfo properties.

In addition, you can get a signing time using the PdfSignature.SigningTime property. This property is used to validate the signature.

Save a signed document.

To accomplish this task, call the [PdfDocumentProcessor.SaveDocument](#) method and pass the PdfSaveOptions object containing a signature as a parameter.

Example

The following example shows how to sign a document using the PDF Document API: [How to: Add a Digital Signature into a PDF Document](#)

See Also

[Protecting a Document](#)

Examples

[Office File API](#) > [PDF Document API](#) > [Examples](#)

This section provides a list of examples (grouped by features) contained in this help.

Document Creation API

- [How to: Generate a Document Layout from Scratch](#)
- [How to: Create Graphics in a Document with Landscape and Portrait Page Orientations](#)
- [How to: Add a Link to a Page](#)
- [How to: Add a Link to URI](#)
- [How to: Add Bookmarks to a Document](#)
- [How to: Bookmark Search Results in a Document](#)
- [How to: Create an Interactive Form](#)
- [How to: Attach a File to a Document](#)

Interactive Form

- [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#)
- [How to: Obtain a Checked Appearance Name for a Check Box](#)
- [How to: Fill an Interactive Form with Values](#)
- [How to: Export Interactive Form Data to XML](#)
- [How to: Import Interactive Form Data from XML](#)
- [How to: Flatten Interactive Form Fields in a Document](#)
- [How to: Add Interactive Form Fields to a PDF Document](#)
- [How to: Delete Interactive Form Fields from a PDF Document](#)

Text Markup Annotation

- [How to: Add a Text Markup Annotation to a PDF Document](#)
- [How to: Modify an Existing Text Markup Annotation](#)
- [How to: Delete Particular Markup Annotations from a Page](#)

Extract Content from a PDF Document

- [How to: Extract Images from a Document](#)
- [How to: Extract Text from a Document](#)
- [How to: Search Text in a Document](#)

Manage Pages of a PDF Document

- [How to: Extract Pages from a Document](#)
- [How to: Delete Pages from a Document](#)
- [How to: Merge Documents into a Single PDF File](#)
- [How to: Rotate Pages](#)

Document Protection

- [How to: Protect a PDF Document with a Password](#)
- [How to: Add a Digital Signature into a PDF Document](#)

Printing

- [How to: Use the PDF Printer Settings](#)
- [How to: Customize PDF Print Output](#)

Export a PDF Document to an Image

- [How to: Export a PDF Document to a Bitmap](#)

- [How to: Export a PDF Document to a Multi-Page Tiff](#)

Document Creation API

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#)

This section contains the following examples:

- [How to: Generate a Document Layout from Scratch](#)
- [How to: Create Graphics in a Document with Landscape and Portrait Page Orientations](#)
- [How to: Add a Link to a Page](#)
- [How to: Add a Link to URI](#)
- [How to: Add Bookmarks to a Document](#)
- [How to: Bookmark Search Results in a Document](#)
- [How to: Create an Interactive Form](#)
- [How to: Attach a File to a Document](#)

How to: Generate a Document Layout from Scratch

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Generate a Document Layout from Scratch](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

To generate a document using the Document Creation API:

- Create an empty document with no pages by calling one of the [PdfDocumentProcessor.CreateEmptyDocument](#) overload methods (e.g., using a file path).
- Create PDF graphics represented by an instance of the PdfGraphics class, calling the [PdfDocumentProcessor.CreateGraphics](#) method. To access PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
- Render a page with created graphics by calling the [PdfDocumentProcessor.RenderNewPage](#) method.

C#


(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
 class Program {
 static void Main(string[] args) {
 using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
 // Create an empty document.
 processor.CreateEmptyDocument("../\\..\\Result.pdf");
 // Create and draw PDF graphics.
 using (PdfGraphics graph = processor.CreateGraphics()) {
 DrawGraphics(graph);
 // Render a page with graphics.
 processor.RenderNewPage(PdfPaperSize.Letter, graph);
 }
 }
 }
 static void DrawGraphics(PdfGraphics graph) {
 // Draw text lines on the page.
 SolidBrush black = (SolidBrush)Brushes.Black;
 using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
 graph.DrawString("PDF Document Processor", font1, black, 180, 150);
 }
 using (Font font2 = new Font("Arial", 20)) {
 graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
 }
 using (Font font3 = new Font("Arial", 10)) {
 graph.DrawString("The PDF Document Processor is a non-visual component " +
 "that provides the application programming interface of the PDF Viewer.",
 font3, black, 180, 300);
 }
 }
 }
}

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 1
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents"
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visu
            End Using
        End Sub
    End Class
End Namespace
```


See Also
[Getting Started](#)

How to: Create Graphics in a Document with Landscape and Portrait Page Ori

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Create Graphics in a Document with Landscape and Portrait Page Orientations](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T383767>.

This example shows how to add text to the top left and bottom right of a page in a document with landscape and portrait pages.

PDF graphics are represented by an instance of the PdfGraphics class. To create graphics, call the [PdfDocumentProcessor.CreateGraphics](#) method. To access PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.

To rotate text that should be drawn in a document with landscape and portrait pages, call the PdfGraphics.RotateTransform method.

To move rotated text to desired position (bottom right) on a page, call the PdfGraphics.TranslateTransform method.

To draw text on a page, call the PdfGraphics.DrawString method with the specified text, font, brush and location.

To add graphics to a page foreground, call the PdfGraphics.AddToPageForeground method.

C#

```

(Program.cs)
using System.Drawing;
using System.Collections.Generic;
using DevExpress.Pdf;
namespace CreateGraphics {
    class Program {
        const float DrawingDpi = 72f;
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                processor.LoadDocument("../\\..\\RotatedDocument.pdf");
                using (SolidBrush textBrush = new SolidBrush(Color.FromArgb(100, Color.Blue)))
                    AddGraphics(processor, "text", textBrush);
                processor.SaveDocument("../\\..\\RotatedDocumentWithGraphics.pdf");
            }
        }
        static void AddGraphics(PdfDocumentProcessor processor, string text, SolidBrush textBrush) {
            IList<PdfPage> pages = processor.Document.Pages;
            for (int i = 0; i < pages.Count; i++) {
                PdfPage page = pages[i];
                using (PdfGraphics graphics = processor.CreateGraphics()) {
                   .SizeF actualPageSize = PrepareGraphics(page, graphics);
                    using (Font font = new Font("Segoe UI", 20, FontStyle.Regular)) {
                       .SizeF textSize = graphics.MeasureString(text, font, PdfStringFormat.GenericDefault);
                        PointF topLeft = new PointF(0, 0);
                        PointF bottomRight = new PointF(actualPageSize.Width - textSize.Width, actualPageHeight - textSize.Height);
                        graphics.DrawString(text, font, textBrush, topLeft);
                        graphics.DrawString(text, font, textBrush, bottomRight);
                        graphics.AddToPageForeground(page, DrawingDpi, DrawingDpi);
                    }
                }
            }
        }
        static SizeF PrepareGraphics(PdfPage page, PdfGraphics graphics) {
            PdfRectangle cropBox = page.CropBox;
            float cropBoxWidth = (float)cropBox.Width;
            float cropBoxHeight = (float)cropBox.Height;
            switch (page.Rotate) {
                case 90:
                    graphics.RotateTransform(-90);
                    graphics.TranslateTransform(-cropBoxHeight, 0);
                    return new SizeF(cropBoxHeight, cropBoxWidth);
                case 180:
                    graphics.RotateTransform(-180);
                    graphics.TranslateTransform(-cropBoxWidth, -cropBoxHeight);
                    return new SizeF(cropBoxWidth, cropBoxHeight);
                case 270:
                    graphics.RotateTransform(-270);
                    graphics.TranslateTransform(0, -cropBoxWidth);
                    return new SizeF(cropBoxHeight, cropBoxWidth);
            }
            return new SizeF(cropBoxWidth, cropBoxHeight);
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports System.Drawing
Imports System.Collections.Generic
Imports DevExpress.Pdf
Namespace CreateGraphics
    Friend Class Program
        Private Const DrawingDpi As Single = 72F
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                processor.LoadDocument("../..\RotatedDocument.pdf")
                Using textBrush As New SolidBrush(Color.FromArgb(100, Color.Red))
                    AddGraphics(processor, "text", textBrush)
                End Using
                processor.SaveDocument("../..\RotatedDocumentWithGraphics.pdf")
            End Using
        End Sub
        Private Shared Sub AddGraphics(ByVal processor As PdfDocumentProcessor, ByVal text As String)
            Dim pages As IList(Of PdfPage) = processor.Document.Pages
            For i As Integer = 0 To pages.Count - 1
                Dim page As PdfPage = pages(i)
                Using graphics As PdfGraphics = processor.CreateGraphics()
                    Dim actualPageSize As.SizeF = PrepareGraphics(page, graphics)
                    Using font As New Font("Segoe UI", 20, FontStyle.Regular)
                        Dim textSize As.SizeF = graphics.MeasureString(text, font)
                        Dim topLeft As New PointF(0, 0)
                        Dim bottomRight As New PointF(actualPageSize.Width, actualPageSize.Height)
                        graphics.DrawString(text, font, textBrush, topLeft)
                        graphics.DrawString(text, font, textBrush, bottomRight)
                        graphics.AddToPageForeground(page, DrawingDpi, DrawOrder.Last)
                    End Using
                End Using
            Next i
        End Sub
        Private Shared Function PrepareGraphics(ByVal page As PdfPage, ByVal graphics As PdfGraphics) As.SizeF
            Dim cropBox As PdfRectangle = page.CropBox
            Dim cropBoxWidth As Single = CSng(cropBox.Width)
            Dim cropBoxHeight As Single = CSng(cropBox.Height)
            Select Case page.Rotate
                Case 90
                    graphics.RotateTransform(-90)
                    graphics.TranslateTransform(-cropBoxHeight, 0)
                    Return New.SizeF(cropBoxHeight, cropBoxWidth)
                Case 180
                    graphics.RotateTransform(-180)
                    graphics.TranslateTransform(-cropBoxWidth, -cropBoxHeight)
                    Return New.SizeF(cropBoxWidth, cropBoxHeight)
                Case 270
                    graphics.RotateTransform(-270)
                    graphics.TranslateTransform(0, -cropBoxWidth)
                    Return New.SizeF(cropBoxHeight, cropBoxWidth)
            End Select
        End Function
    End Class
End Namespace


```

```
        Return New SizeF(cropBoxWidth, cropBoxHeight)
    End Function
End Class
End Namespace
```


See Also[Getting Started](#)

How to: Add a Link to a Page

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Add a Link to a Page](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494761>.

This example shows how to add a link to a page using the PDF Document Creation API.

 **Note**

The measurement unit of X, Y destination coordinates is equal to **1/72** of an inch.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System.Drawing;
namespace AddLinkToPage {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("../\\..\\Result.pdf");
                // Create and draw graphics.
                using (PdfGraphics graphics = processor.CreateGraphics()) {
                    DrawGraphics(graphics);
                    // Create a link to a page specifying link area, the page number and X, Y destinations.
                    graphics.AddLinkToPage(new RectangleF(180, 160, 480, 30), 1, 168, 230);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graphics) {
            // Draw a text line on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graphics.DrawString("PDF Document Processor", font, black, 180, 150);
            }
        }
    }
}
```

Visual Basic


```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Drawing
Namespace AddLinkToPage
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw graphics.
                Using graphics As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graphics)
                    ' Create a link to a page specifying link area, the pa
                    graphics.AddLinkToPage(New RectangleF(180, 160, 480, 3
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graphics As PdfGraphics)
            ' Draw a text line on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font As New Font("Times New Roman", 32, FontStyle.Bold)
                graphics.DrawString("PDF Document Processor", font, black,
            End Using
        End Sub
    End Class
End Namespace
```

See Also


[Getting Started](#)

How to: Add a Link to URI

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Add a Link to URI](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T495074>.

This example shows how to add a hyperlink to a URI using the PdfGraphics.AddLinkToUri method.

C#

```
(Program.cs)
using System;
using DevExpress.Pdf;
using System.Drawing;
namespace AddLinkToUri {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("../\\..\\Result.pdf");
                // Create and draw graphics.
                using (PdfGraphics graphics = processor.CreateGraphics()) {
                    DrawGraphics(graphics);
                    // Create a link to URI specifying link area and URI.
                    graphics.AddLinkToUri(new RectangleF(310, 150, 180, 15), new Uri("https://www.devexpress.com"));
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graphics) {
            // Draw a text line on the page.
            using (Font font = new Font("Arial", 10)) {
                SolidBrush blue = (SolidBrush)Brushes.Blue;
                graphics.DrawString("https://www.devexpress.com", font, blue, 310, 150);
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports System
Imports DevExpress.Pdf
Imports System.Drawing
Namespace AddLinkToUri
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw graphics.
                Using graphics As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graphics)
                    ' Create a link to URI specifying link area and URI.
                    graphics.AddLinkToUri(New RectangleF(310, 150, 180, 15
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graphics As PdfGraphics)
            ' Draw a text line on the page.
            Using font As New Font("Arial", 10)
                Dim blue As SolidBrush = CType(Brushes.Blue, SolidBrush)
                graphics.DrawString("https://www.devexpress.com", font, bl
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

How to: Add Bookmarks to a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Add Bookmarks to a Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T495077>.

This example shows how to create bookmarks in code and add them to a document.

To do this:

- Create a PdfBookmark object;
- Specify the bookmark title and destination using the PdfBookmark.Title and PdfBookmark.Destination properties. To create a destination, call a [PdfDocumentProcessor.CreateDestination](#) overloaded method.
- Add the bookmark to the bookmarks collection, which is accessed from the PdfDocument.Bookmarks property.

Note

The measurement unit in the destination is equal to **1/72** of an inch.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace AddBookmarks {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\Document.pdf");
                // Create bookmarks and add them to the PDF document.
                PdfDestination destination1 = processor.CreateDestination(1, 180, 150);
                PdfDestination destination2 = processor.CreateDestination(1, 168, 230);
                PdfDestination destination3 = processor.CreateDestination(1, 20, 350);
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "PDF Document Processor", Destination = destination1 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Display, Print and Export PDF", Destination = destination2 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Learn More", Destination = destination3 });
                // Save the result document.
                processor.SaveDocument("..\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic


```
(Program.vb)
Imports DevExpress.Pdf
Namespace AddBookmarks
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Create bookmarks and add them to the PDF document.
                Dim destination1 As PdfDestination = processor.CreateDesti
                Dim destination2 As PdfDestination = processor.CreateDesti
                Dim destination3 As PdfDestination = processor.CreateDesti
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.
                ' Save the result document.
                processor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also


[Getting Started](#)

How to: Bookmark Search Results in a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Bookmark Search Results in a Document](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T350881>.

This example shows how to create a bookmark with a destination corresponding to the found text in a document.

C#

```
(Program.cs)
using System.Collections.Generic;
using DevExpress.Pdf;
namespace Destination {
    class Program {
        static void Main(string[] args) {
            // Create a PDF document processor.
            using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
                // Define search words.
                string[] words = { "DX-B5000", "DX-RX800" };
                // Load a PDF document.
                documentProcessor.LoadDocument(@"..\..\Document.pdf");
                // Specify the search parameters.
                PdfTextSearchParameters searchParameters = new PdfTextSearchParameters();
                searchParameters.CaseSensitive = true;
                searchParameters.WholeWords = true;
                // Get bookmark list.
                IList<PdfBookmark> bookmarks = documentProcessor.Document.Bookmarks;
                foreach (string word in words) {
                    // Get the search results from the FindText method called with search text and search
                    PdfTextSearchResults results = documentProcessor.FindText(word, searchParameters);
                    // If the desired text is found, create a destination that corresponds to the found text
                    // to be displayed at the upper corner of the window.
                    if (results.Status == PdfTextSearchStatus.Found) {
                        PdfXYZDestination destination = new PdfXYZDestination(results.Page, 0, results.Rect);
                        // Create a bookmark with the search word shown as title and the destination.
                        PdfBookmark bookmark = new PdfBookmark() { Title = word, Destination = destination };
                        // Add the bookmark to the bookmark list.
                        bookmarks.Add(bookmark);
                    }
                }
                // Save the modified document.
                documentProcessor.SaveDocument(@"..\..\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```

(Program.vb)
Imports System.Collections.Generic
Imports DevExpress.Pdf
Namespace Destination
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF document processor.
            Using documentProcessor As New PdfDocumentProcessor()
                ' Define search words.
                Dim words() As String = { "DX-B5000", "DX-RX800" }
                ' Load a PDF document.
                documentProcessor.LoadDocument("../..\Document.pdf")
                ' Specify the search parameters.
                Dim searchParameters As New PdfTextSearchParameters()
                searchParameters.CaseSensitive = True
                searchParameters.WholeWords = True
                ' Get bookmark list.
                Dim bookmarks As IList(Of PdfBookmark) = documentProcessor
                For Each word As String In words
                    ' Get the search results from the FindText method call
                    Dim results As PdfTextSearchResults = documentProcesso
                    ' If the desired text is found, create a destination t
                    ' to be displayed at the upper corner of the window.
                    If results.Status = PdfTextSearchStatus.Found Then
                        Dim destination As New PdfXYZDestination(results.P
                        ' Create a bookmark with the search word shown as
                        Dim bookmark As New PdfBookmark() With {.Title = w
                        ' Add the bookmark to the bookmark list.
                        bookmarks.Add(bookmark)
                    End If
                Next word
                ' Save the modified document.
                documentProcessor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace

```

See Also
[Getting Started](#)

How to: Highlight Search Results in a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Highlight Search Results in a Document](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T568675>.

This example shows how to highlight search results in a document by drawing filled rectangles that correspond to the document area containing found text.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System.Drawing;
namespace HighlightSearchResults {
    class Program {
        static void Main(string[] args) {
            // Create a PDF document processor.
            using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
                // Define search words.
                string[] words = { "Get", "DX-RX809", "HD", "DX-B5000" };
                // Load a PDF document.
                documentProcessor.LoadDocument(@"..\..\Document.pdf");
                // Specify the search parameters.
                PdfTextSearchParameters searchParameters = new PdfTextSearchParameters();
                searchParameters.CaseSensitive = true;
                searchParameters.WholeWords = true;
                foreach (string word in words) {
                    // Get the search results from the FindText method called with search text and search
                    PdfTextSearchResults result = documentProcessor.FindText(word, searchParameters);
                    // If the desired text is found, create a rectangle that corresponds to the area conta
                    while (result.Status == PdfTextSearchStatus.Found) {
                        using (PdfGraphics graphics = documentProcessor.CreateGraphics()) {
                            HighlightFoundText(graphics, result, new SolidBrush(Color.FromArgb(130, 55, 15)));
                        }
                        result = documentProcessor.FindText(word, searchParameters);
                    }
                }
                // Save the modified document.
                documentProcessor.SaveDocument(@"..\..\Result.pdf");
            }
        }
        public static void HighlightFoundText(PdfGraphics graphics, PdfTextSearchResults result, SolidBrush brush) {
            for (int i = 0; i < result.Rectangles.Count; i++) {
                RectangleF rect = new RectangleF(new PointF((float)result.Rectangles[i].Left, (float)result.Rectangles[i].Top),
                    new SizeF((float)result.Rectangles[i].Width, (float)result.Rectangles[i].Height));
                graphics.FillRectangle(brush, rect);
            }
            graphics.AddToPageForeground(result.Page, 72, 72);
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Drawing
Namespace HighlightSearchResults
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF document processor.
            Using documentProcessor As New PdfDocumentProcessor()
                ' Define search words.
                Dim words() As String = { "Get", "DX-RX809", "HD", "DX-B50"
                ' Load a PDF document.
                documentProcessor.LoadDocument("..\\..\\Document.pdf")
                ' Specify the search parameters.
                Dim searchParameters As New PdfTextSearchParameters()
                searchParameters.CaseSensitive = True
                searchParameters.WholeWords = True
                For Each word As String In words
                    ' Get the search results from the FindText method call
                    Dim result As PdfTextSearchResults = documentProcessor
                    ' If the desired text is found, create a rectangle tha
                    Do While result.Status = PdfTextSearchStatus.Found
                        Using graphics As PdfGraphics = documentProcessor.
                            HighlightFoundText(graphics, result, New Solid
                        End Using
                        result = documentProcessor.FindText(word, searchPa
                    Loop
                Next word
                ' Save the modified document.
                documentProcessor.SaveDocument("..\\..\\Result.pdf")
            End Using
        End Sub
        Public Shared Sub HighlightFoundText(ByVal graphics As PdfGraphics
            For i As Integer = 0 To result.Rectangles.Count - 1
                Dim rect As New RectangleF(New PointF(CSng(result.Rectangl
                graphics.FillRectangle(brush, rect)
            Next i
            graphics.AddToPageForeground(result.Page, 72, 72)
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

How to: Create an Interactive Form

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Create an Interactive Form](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494635>.

This example shows how to add interactive form fields (e.g., text box and radio button group fields) to a PDF document using a PdfGraphics object.

To access the PdfGraphics, you need to reference the DevExpress.Pdf.Drawing assembly.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System.Drawing;
namespace AddFormFieldsToNewDocument {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create graphics and draw form fields.
                using (PdfGraphics graphics = processor.CreateGraphics()) {
                    DrawFormFields(graphics);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics);
                }
            }
        }
        static void DrawFormFields(PdfGraphics graphics) {
            // Create a text box field and specify its location on the page.
            PdfGraphicsAcroFormTextBoxField textBox = new PdfGraphicsAcroFormTextBoxField("text box", new
            // Specify text box text, and appearance.
            textBox.Text = "Text Box";
            textBox.Appearance.FontSize = 12;
            textBox.Appearance.BackgroundColor = Color.AliceBlue;
            // Add the text box field to graphics.
            graphics.AddFormField(textBox);
            // Create a radio group field.
            PdfGraphicsAcroFormRadioGroupField radioGroup = new PdfGraphicsAcroFormRadioGroupField("First
            // Add the first radio button to the group and specify its location using a RectangleF object
            radioGroup.AddButton("button1", new RectangleF(30, 60, 20, 20));
            // Add the second radio button to the group.
            radioGroup.AddButton("button2", new RectangleF(30, 90, 20, 20));
            // Specify radio group selected index, and appearance.
            radioGroup.SelectedIndex = 0;
            radioGroup.Appearance.BorderAppearance = new PdfGraphicsAcroFormBorderAppearance() { Color = C
            // Add the radio group field to graphics.
            graphics.AddFormField(radioGroup);
        }
    }
}
```

Visual Basic


```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Drawing
Namespace AddFormFieldsToNewDocument
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create graphics and draw form fields.
                Using graphics As PdfGraphics = processor.CreateGraphics()
                    DrawFormFields(graphics)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graphics)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawFormFields(ByVal graphics As PdfGraphics)
            ' Create a text box field and specify its location on the page
            Dim textBox As New PdfGraphicsAcroFormTextBoxField("text box",
            ' Specify text box text, and appearance.
            textBox.Text = "Text Box"
            textBox.Appearance.FontSize = 12
            textBox.Appearance.BackgroundColor = Color.AliceBlue
            ' Add the text box field to graphics.
            graphics.AddFormField(textBox)
            ' Create a radio group field.
            Dim radioGroup As New PdfGraphicsAcroFormRadioGroupField("Firs
            ' Add the first radio button to the group and specify its loca
            radioGroup.AddButton("button1", New RectangleF(30, 60, 20, 20)
            ' Add the second radio button to the group.
            radioGroup.AddButton("button2", New RectangleF(30, 90, 20, 20)
            ' Specify radio group selected index, and appearance.
            radioGroup.SelectedIndex = 0
            radioGroup.Appearance.BorderAppearance = New PdfGraphicsAcroFo
            ' Add the radio group field to graphics.
            graphics.AddFormField(radioGroup)
        End Sub
    End Class
End Namespace
```

See Also


[Getting Started](#)

How to: Attach a File to a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Creation API](#) > [How to: Attach a File to a Document](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T304234>.

This example shows how to programmatically attach a file to the PDF document.

- To do this:
- Create a PdfFileAttachment object;
 - Specify the attachment creation date, description, and file name using PdfFileAttachment.CreationDate, PdfFileAttachment.Description, and PdfFileAttachment.FileName properties. To specify the data for the attachment, use the PdfFileAttachment.Data property;
 - If required, you can specify additional properties of an attached file, for example, the file's mime type and relationship using PdfFileAttachment.MimeType and PdfFileAttachment.Relationship properties, respectively.
 - Call the [PdfDocumentProcessor.AttachFile](#) method with a file attachment object used as a parameter.
 - Save the attachment to a document by calling the [PdfDocumentProcessor.SaveDocument](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.IO;
namespace AttachFile {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\..\\Document.pdf");
                // Attach a file to the PDF document.
                processor.AttachFile(new PdfFileAttachment() {
                    CreationDate = DateTime.Now,
                    Description = "This is my attach file.",
                    FileName = "MyAttach.txt",
                    Data = File.ReadAllBytes("..\\..\\..\\FileToAttach.txt")
                });
                // The attached document.
                processor.SaveDocument("..\\..\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.IO
Namespace AttachFile
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("..\\..\\Document.pdf")
                ' Attach a file to the PDF document.
                processor.AttachFile(New PdfFileAttachment() With { _
                    .CreationDate = Date.Now, _
                    .Description = "This is my attach file.", _
                    .FileName = "MyAttach.txt", _
                    .Data = File.ReadAllBytes("..\\..\\FileToAttach.txt") _
                })
                ' The attached document.
                processor.SaveDocument("..\\..\\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

Interactive Form

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#)

This section contains the following examples:

- [How to: Obtain a Checked Appearance Name for a Check Box](#)
- [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#)
- [How to: Fill an Interactive Form with Values](#)
- [How to: Export Interactive Form Data to XML](#)
- [How to: Import Interactive Form Data from XML](#)
- [How to: Flatten Interactive Form Fields in a Document](#)
- [How to: Add Interactive Form Fields to a PDF Document](#)
- [How to: Delete Interactive Form Fields from a PDF Document](#)

How to: Obtain a Checked Appearance Name for Each Radio Button in the Ra

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#) > [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T622985>.

This example shows how to get a checked appearance name for each radio button and check the corresponding radio button with the obtained value.

To accomplish this task, call the **GetRadioGroupCheckedAppearanceNames** method using a [PdfDocumentProcessor](#) instance and the field.

To obtain interactive form data, call the [PdfDocumentProcessor.GetFormData](#) method. To check the "Female" radio button, use the PdfFormData.Value property.

C#

```

(Program.cs)
using System.Collections.Generic;
using DevExpress.Pdf;
using System.Linq;
namespace GetRadioGroupCheckedValues {
    class Program {
        static void Main(string[] args) {
            // Load a document with an interactive form.
            PdfDocumentProcessor processor = new PdfDocumentProcessor();
            processor.LoadDocument(@"DocumentToFill.pdf");
            // Obtain interactive form data from a document.
            PdfForm data = processor.GetForm data();
            // Obtain the radio group checked appearance names.
            List<string> radioGroupCheckedValues = GetRadioGroupCheckedAppearanceNames(processor, "Gender");
            // Specify a checked appearance name for the Female radio button.
            form data["Gender"].Value = radioGroupCheckedValues[0];
            // Apply data to the interactive form.
            processor.ApplyForm data(form data);
            // Save the modified document.
            processor.SaveDocument("../..\\Result.pdf");
        }
        static List<string> GetRadioGroupCheckedAppearanceNames(PdfDocumentProcessor processor, string fieldName) {
            if (processor.Document == null || processor.Document.AcroForm == null)
                return null;
            PdfInteractiveFormField button = FindRadioButton(processor.Document.AcroForm.Fields, "", fieldName);
            if (button != null) {
                List<string> result = new List<string>();
                foreach (PdfInteractiveFormField kid in button.Kids) {
                    if (kid.Widget != null && kid.Widget.Appearance != null) {
                        List<string> names = kid.Widget.Appearance.Down.Forms.Keys.ToList();
                        if (names.Count == 1)
                            result.Add(names[0]);
                        if (names.Count > 1) {
                            if (names[0] == "Off")
                                result.Add(names[1]);
                            else result.Add(names[0]);
                        }
                    }
                }
                return result;
            }
            return null;
        }
        static PdfInteractiveFormField FindRadioButton(IEnumerable<PdfInteractiveFormField> fields, string partialName, string fieldName) {
            foreach (PdfInteractiveFormField field in fields) {
                if (partialName + field.Name == fieldName) {
                    if (field.Flags.HasFlag(PdfInteractiveFormFieldFlags.Radio))
                        return field as PdfButtonFormField;
                    return null;
                }
                if (field.Kids != null) {
                    PdfInteractiveFormField intermediateResult = FindRadioButton(field.Kids, partialName + field.Name + " ", fieldName);
                    if (intermediateResult != null)
                        return intermediateResult;
                }
            }
            return null;
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports System.Collections.Generic
Imports DevExpress.Pdf
Imports System.Linq
Namespace GetRadioGroupCheckedValues
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Load a document with an interactive form.
            Dim processor As New PdfDocumentProcessor()
            processor.LoadDocument("DocumentToFill.pdf")
            ' Obtain interactive form data from a document.
            Dim formData As PdfFormData = processor.GetFormData()
            ' Obtain the radio group checked appearance names.
            Dim radioGroupCheckedValues As List(Of String) = GetRadioGroup
            ' Specify a checked appearance name for the Female radio butto
            formData("Gender").Value = radioGroupCheckedValues(0)
            ' Apply data to the interactive form.
            processor.ApplyFormData(formData)
            ' Save the modified document.
            processor.SaveDocument("../..\Result.pdf")
        End Sub
        Private Shared Function GetRadioGroupCheckedAppearanceNames(ByVal
            If processor.Document Is Nothing OrElse processor.Document.Acr
                Return Nothing
            End If
            Dim button As PdfInteractiveFormField = FindRadioButton(proces
            If button IsNot Nothing Then
                Dim result As New List(Of String)()
                For Each kid As PdfInteractiveFormField In button.Kids
                    If kid.Widget IsNot Nothing AndAlso kid.Widget.Appearan
                        Dim names As List(Of String) = kid.Widget.Appearan
                        If names.Count = 1 Then
                            result.Add(names(0))
                        End If
                        If names.Count > 1 Then
                            If names(0) = "Off" Then
                                result.Add(names(1))
                            Else
                                result.Add(names(0))
                            End If
                        End If
                    End If
                End If
                Return result
            End If
            Return Nothing
        End Function
        Private Shared Function FindRadioButton(ByVal fields As IEnumerable
            For Each field As PdfInteractiveFormField In fields
                If partialName & field.Name = fieldName Then
                    If field.Flags.HasFlag(PdfInteractiveFormFieldFlags.Ra

```

```
        Return TryCast(field, PdfButtonFormField)
    End If
    Return Nothing
End If
If field.Kids IsNot Nothing Then
    Dim intermediateResult As PdfInteractiveFormField = Fi
    If intermediateResult IsNot Nothing Then
        Return intermediateResult
    End If
End If
Next field
Return Nothing
End Function
End Class
End Namespace
```

How to: Obtain a Checked Appearance Name for a Check Box

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#) > [How to: Obtain a Checked Appearance Name for a Check Box](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T609857>.

This example shows how to get a checked appearance name for the "Female" check box and check the check box with this value.

To accomplish this task, call the **GetCheckboxCheckedValue** method using a [PdfDocumentProcessor](#) instance and the field name.

To obtain interactive form data, call the [PdfDocumentProcessor.GetFormData](#) method. To check the "Female" check box, use the PdfFormData.Value property.

C#

```

(Program.cs)
using System.Collections.Generic;
using System.Linq;
using DevExpress.Pdf;
namespace GetCheckboxCheckedValue {
    class Program {
        static void Main(string[] args) {
            // Load a document with an interactive form.
            PdfDocumentProcessor processor = new PdfDocumentProcessor();
            processor.LoadDocument("../..\\InteractiveForm.pdf");
            // Obtain interactive form data from a document.
            PdfFormData formData = processor.GetFormData();
            // Specify a checked value for the Female form field obtained from the GetCheckboxCheckedValue
            formData["Female"].Value = GetCheckboxCheckedValue(processor, "Female");
            // Apply data to the interactive form.
            processor.ApplyFormData(formData);
            // Save the modified document.
            processor.SaveDocument("../..\\Result.pdf");
        }
        static string GetCheckboxCheckedValue(PdfDocumentProcessor processor, string fieldName) {
            if (processor.Document == null || processor.Document.AcroForm == null)
                return null;
            PdfInteractiveFormField button = FindCheckBox(processor.Document.AcroForm.Fields, "", fieldName);
            if (button != null) {
                PdfAnnotationAppearances appearance = button.Widget.Appearance;
                if (appearance != null) {
                    IList<string> names = appearance.Normal.Forms.Keys.ToList();
                    if (names.Count == 1)
                        return names.First();
                    if (names.Count > 1)
                        return names[0] == "Off" ? names[1] : names[0];
                }
                return null;
            }
            return null;
        }
        static PdfInteractiveFormField FindCheckBox(IEnumerable<PdfInteractiveFormField> fields, string partialName, string fieldName) {
            foreach (PdfInteractiveFormField field in fields) {
                if (partialName + field.Name == fieldName) {
                    if (field.Flags.HasFlag(PdfInteractiveFormFieldFlags.Radio) || field.Flags.HasFlag(PdfInteractiveFormFieldFlags.Button))
                        return field as PdfButtonFormField;
                }
                if (field.Kids != null) {
                    PdfInteractiveFormField intermediateResult = FindCheckBox(field.Kids, partialName + " ", fieldName);
                    if (intermediateResult != null)
                        return intermediateResult;
                }
            }
            return null;
        }
    }
}

```

Visual Basic

```


(Program.vb)
Imports System.Collections.Generic
Imports System.Linq
Imports DevExpress.Pdf
Namespace GetCheckboxCheckedValue
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Load a document with an interactive form.
            Dim processor As New PdfDocumentProcessor()
            processor.LoadDocument("../..\InteractiveForm.pdf")
            ' Obtain interactive form data from a document.
            Dim formData As PdfFormData = processor.GetFormData()
            ' Specify a checked value for the Female form field obtained f
            formData("Female").Value = GetCheckboxCheckedValue(processor,
            ' Apply data to the interactive form.
            processor.ApplyFormData(formData)
            ' Save the modified document.
            processor.SaveDocument("../..\Result.pdf")
        End Sub
        Private Shared Function GetCheckboxCheckedValue(ByVal processor As
            If processor.Document Is Nothing OrElse processor.Document.Acr
                Return Nothing
            End If
            Dim button As PdfInteractiveFormField = FindCheckBox(processor
            If button IsNot Nothing Then
                Dim appearance As PdfAnnotationAppearances = button.Widget
                If appearance IsNot Nothing Then
                    Dim names As IList(Of String) = appearance.Normal.Form
                    If names.Count = 1 Then
                        Return names.First()
                    End If
                    If names.Count > 1 Then
                        Return If(names(0) = "Off", names(1), names(0))
                    End If
                End If
                Return Nothing
            End If
            Return Nothing
        End Function
        Private Shared Function FindCheckBox(ByVal fields As IEnumerable(C
            For Each field As PdfInteractiveFormField In fields
                If partialName & field.Name = fieldName Then
                    If field.Flags.HasFlag(PdfInteractiveFormFieldFlags.Ra
                        Return Nothing
                    End If
                    Return TryCast(field, PdfButtonFormField)
                End If
                If field.Kids IsNot Nothing Then
                    Dim intermediateResult As PdfInteractiveFormField = Fi
                    If intermediateResult IsNot Nothing Then
                        Return intermediateResult
                    End If
                End If
            Next
        End Function
    End Class
End Namespace

```


```
        End If
    End If
    Next field
    Return Nothing
End Function
End Class
End Namespace
```

How to: Fill an Interactive Form with Values

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#) > [How to: Fill an Interactive Form with Values](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [PdfDocumentProcessor.GetFormData](#) method. Then, specify a value for a form field using the PdfFormData.Value property.

To obtain the names of the interactive form fields, use the [PdfDocumentProcessor.GetFormFieldNames](#) method.

To apply data to the interactive form, call the [PdfDocumentProcessor.ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#


```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic


```
(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals",
    ' Obtain data from the Address form field and specify values for Address
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using
```

How to: Export Interactive Form Data to XML

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#) > [How to: Export Interactive Form Data to XML](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T491302>.

This example shows how to export AcroForm data (interactive form data) from a PDF document to XML format. You can also export the AcroForm data to FDF, XFDF, and TXT formats.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace ExportInteractiveForms {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a PDF document with AcroForm data.
                processor.LoadDocument("../..\..\InteractiveForm.pdf");
                // Export AcroForm data to XML format.
                processor.Export("../..\..\InteractiveForm.xml", PdfFormDataType.Xml);
            }
        }
    }
}
```


Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace ExportInteractiveForms
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a PDF document with AcroForm data.
                processor.LoadDocument("../..\..\InteractiveForm.pdf")
                ' Export AcroForm data to XML format.
                processor.Export("../..\..\InteractiveForm.xml", PdfFormDataType.Xml)
            End Using
        End Sub
    End Class
End Namespace
```


See Also
[Getting Started](#)

How to: Import Interactive Form Data from XML

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#) > [How to: Import Interactive Form Data from XML](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T491383>.

This example demonstrates how you can import AcroForm data (interactive forms data) from XML format to a PDF document. You can also import the AcroForm data from FDF, XFDF, and TXT formats.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace ImportInteractiveForms {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a PDF document with AcroForm data.
                processor.LoadDocument("..\\..\\EmptyForm.pdf");
                // Import AcroForm data from an XML file.
                processor.Import("..\\..\\InteractiveForm.xml");
                // Save the imported document.
                processor.SaveDocument("..\\..\\InteractiveForm.pdf");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace ImportInteractiveForms
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a PDF document with AcroForm data.
                processor.LoadDocument("..\\..\\EmptyForm.pdf")
                ' Import AcroForm data from an XML file.
                processor.Import("..\\..\\InteractiveForm.xml")
                ' Save the imported document.
                processor.SaveDocument("..\\..\\InteractiveForm.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also
[Getting Started](#)

How to: Flatten Interactive Form Fields in a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#) > [How to: Flatten Interactive Form Fields in a Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T446039>.

This example shows how to flatten interactive form fields (e.g., text fields, list boxes).

To flatten a whole interactive form, call the [PdfDocumentProcessor.FlattenForm](#) method. If the interactive form is flattened successfully, this method returns **true**. If the document doesn't contain an interactive form that should be flattened, this method returns **false**.

To flatten a particular form field by its name, call the [PdfDocumentProcessor.FlattenFormField](#) method with a field name used as a parameter. This method tries to find the interactive form field in a document using the field name. If this form field is found in the document, then it will be flattened and this method returns **true**; otherwise, it returns **false**.

C#

(Program.cs)
using DevExpress.Pdf;
using System;
namespace FlattenInteractiveForm {
 class Program {
 static void Main(string[] args) {
 using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
 // Load a document with an interactive form.
 processor.LoadDocument("../..\\Document.pdf");
 // Flatten a form field by its name
 if (processor.FlattenFormField("Nationality"))
 // Save a document with the flattened form field.
 processor.SaveDocument("../..\\Result1.pdf");
 // Show a message if the form field was not found in a document.
 else
 Console.WriteLine("A document does not contain a form field with the specified name to be flattened.");
 // Flatten a whole interactive form.
 if (processor.FlattenForm())
 // Save a document with the flattened form.
 processor.SaveDocument("../..\\Result2.pdf");
 // Show a message if the interactive was not found in a document.
 else
 Console.WriteLine("A document does not contain an interactive form to be flattened.");
 }
 }
 }
}

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Namespace FlattenInteractiveForm
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document with an interactive form.
                processor.LoadDocument("../..\Document.pdf")
                ' Flatten a form field by its name
                If processor.FlattenFormField("Nationality") Then
                    ' Save a document with the flattened form field.
                    processor.SaveDocument("../..\Result1.pdf")
                    ' Show a message if the form field was not found in a document
                Else
                    Console.WriteLine("A document does not contain a form field")
                End If
                ' Flatten a whole interactive form.
                If processor.FlattenForm() Then
                    ' Save a document with the flattened form.
                    processor.SaveDocument("../..\Result2.pdf")
                    ' Show a message if the interactive form was not found in a document
                Else
                    Console.WriteLine("A document does not contain an interactive form")
                End If
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

How to: Add Interactive Form Fields to a PDF Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#) > [How to: Add Interactive Form Fields to a PDF Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494642>.

This example shows how to create interactive form fields (e.g., text box and radio button group fields) and add them to a document.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace AddFormFieldsToExistingDocument {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\..\\Document.pdf");
                // Create a text box field specifying the field name, page number, and field location on the page.
                PdfAcroFormTextBoxField textBox = new PdfAcroFormTextBoxField("text box", 1, new PdfRectangle(230, 635, 250, 655));
                // Specify text box text, and appearance.
                textBox.Text = "Text Box";
                textBox.Appearance.BackgroundColor = new PdfRGBColor(0.8, 0.5, 0.3);
                textBox.Appearance.FontSize = 12;
                // Create a radio group field specifying its name and the page number.
                PdfAcroFormRadioGroupField radioGroup = new PdfAcroFormRadioGroupField("Gender Group", 1);
                // Add the first radio button to the group and specify its location using a PdfRectangle object.
                radioGroup.AddButton("button1", new PdfRectangle(230, 635, 250, 655));
                // Add the second radio button to the group.
                radioGroup.AddButton("button2", new PdfRectangle(310, 635, 330, 655));
                // Specify radio group selected index, and appearance.
                radioGroup.SelectedIndex = 0;
                radioGroup.Appearance.BorderAppearance = new PdfAcroFormBorderAppearance() { Color = new PdfRGBColor(0.8, 0.5, 0.3) };
                // Add form fields to the page.
                processor.AddFormFields(textBox, radioGroup);
                // Save the result document.
                processor.SaveDocument("..\\..\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace AddFormFieldsToExistingDocument
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Create a text box field specifying the field name, page
                Dim textBox As New PdfAcroFormTextBoxField("text box", 1,
                ' Specify text box text, and appearance.
                textBox.Text = "Text Box"
                textBox.Appearance.BackgroundColor = New PdfRGBColor(0.8,
                textBox.Appearance.FontSize = 12
                ' Create a radio group field specifying its name and the p
                Dim radioGroup As New PdfAcroFormRadioGroupField("Gender G
                ' Add the first radio button to the group and specify its
                radioGroup.AddButton("button1", New PdfRectangle(230, 635,
                ' Add the second radio button to the group.
                radioGroup.AddButton("button2", New PdfRectangle(310, 635,
                ' Specify radio group selected index, and appearance.
                radioGroup.SelectedIndex = 0
                radioGroup.Appearance.BorderAppearance = New PdfAcroFormBo
                ' Add form fields to the page.
                processor.AddFormFields(textBox, radioGroup)
                ' Save the result document.
                processor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also
[Getting Started](#)

How to: Delete Interactive Form Fields from a PDF Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Interactive Form](#) > [How to: Delete Interactive Form Fields from a PDF Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494240>.

This example shows how to remove a particular form field and a whole interactive form from a PDF document.

To remove all form fields from a document, call the [PdfDocumentProcessor.RemoveForm](#) method.

If the interactive form is removed successfully, this method returns **true**. If the document doesn't contain an interactive form that should be removed, this method returns **false**.

To remove a particular form field by its name, call the [PdfDocumentProcessor.RemoveFormField](#) method and pass a field name as an argument to this method.

The [PdfDocumentProcessor.RemoveFormField](#) method tries to find the field in a document using the field name. If this field is found in the document, then it will be removed from a document and this method returns **true**; otherwise, it returns **false**.

C#

(Program.cs)
using DevExpress.Pdf;
using System;
namespace RemoveInteractiveForm {
 class Program {
 static void Main(string[] args) {
 using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
 // Load a document with an interactive form.
 processor.LoadDocument("../..\\..\\InteractiveForm.pdf");
 // Remove a form field by its name.
 if (processor.RemoveFormField("FirstName"))
 // Save the modified document.
 processor.SaveDocument("../..\\..\\Result1.pdf");
 else
 // Show a message if the form field was not found in a document.
 Console.WriteLine("The form field was not removed from a document. Make sure, the form field name is correct.");
 // Remove the whole interactive form.
 if (processor.RemoveForm())
 // Save the modified document.
 processor.SaveDocument("../..\\..\\Result2.pdf");
 else
 // Show a message if the interactive form was not found in a document.
 Console.WriteLine("The interactive form was not removed from a document. Make sure the document contains an interactive form.");
 }
 }
 }
}

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Namespace RemoveInteractiveForm
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document with an interactive form.
                processor.LoadDocument("../..\InteractiveForm.pdf")
                ' Remove a form field by its name.
                If processor.RemoveFormField("FirstName") Then
                    ' Save the modified document.
                    processor.SaveDocument("../..\Result1.pdf")
                Else
                    ' Show a message if the form field was not found in a
                    Console.WriteLine("The form field was not removed from")
                End If
                ' Remove the whole interactive form.
                If processor.RemoveForm() Then
                    ' Save the modified document.
                    processor.SaveDocument("../..\Result2.pdf")
                Else
                    ' Show a message if the interactive form was not found
                    Console.WriteLine("The interactive form was not remove")
                End If
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

Text Markup Annotation

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Text Markup Annotation](#)

This section contains the following examples:

- [How to: Add a Text Markup Annotation to a PDF Document](#)
- [How to: Modify an Existing Text Markup Annotation](#)
- [How to: Delete Particular Markup Annotations from a Page](#)

How to: Add a Text Markup Annotation to a PDF Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Text Markup Annotation](#) > [How to: Add a Text Markup Annotation to a PDF Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T545395>.

This example shows how to create a text markup annotation that highlights a text in a PDF document and specify the annotation properties.

To add a text markup annotation to a page, call one of [PdfDocumentProcessor.AddTextMarkupAnnotation](#) overload methods, where specify the page number and a page area corresponding to a text that should be annotated on this page. Note that if a specified page area does not correspond to a text on the page, the annotation is not created and [PdfDocumentProcessor.AddTextMarkupAnnotation](#) overload methods return **null**.

C#

(Program.cs)
using System;
using DevExpress.Pdf;
namespace CreateMarkupAnnotation {
 class Program {
 static void Main(string[] args) {
 // Create a PDF Document Processor.
 using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
 // Load a document.
 processor.LoadDocument("../\\..\\Document.pdf");
 // Add a text markup annotation to the first page for a text corresponding to the specified
 PdfTextMarkupAnnotationData annot = processor.AddTextMarkupAnnotation(1, new PdfRectangle(
 PdfTextMarkupAnnotationType.Highlight);

 if (annot != null) {
 // Specify the annotation properties.
 annot.Author = "Bill Smith";
 annot.Contents = "Note";
 annot.Color = new PdfRGBColor(0.8, 0.2, 0.1);
 // Save the result document.
 processor.SaveDocument("../\\..\\Result.pdf");
 }
 else
 Console.WriteLine("The annotation cannot be added to a page. Make sure, a specified page
 }
 }
 }
}

Visual Basic


```
(Program.vb)
Imports System
Imports DevExpress.Pdf
Namespace CreateMarkupAnnotation
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Add a text markup annotation to the first page for a text
                Dim annot As PdfTextMarkupAnnotationData = processor.AddTextMarkupAnnotation(1, "Note")
                If annot IsNot Nothing Then
                    ' Specify the annotation properties.
                    annot.Author = "Bill Smith"
                    annot.Contents = "Note"
                    annot.Color = New PdfRGBColor(0.8, 0.2, 0.1)
                    ' Save the result document.
                    processor.SaveDocument("../..\Result.pdf")
                Else
                    Console.WriteLine("The annotation cannot be added to a PDF document.")
                End If
            End Using
        End Sub
    End Class
End Namespace
```

See Also


[Getting Started](#)

How to: Modify an Existing Text Markup Annotation

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Text Markup Annotation](#) > [How to: Modify an Existing Text Markup Annotation](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T545429>.

This example shows how to change an existing markup annotation's settings in a PDF document.

To retrieve all text markup annotations in a page, call the [PdfDocumentProcessor.GetMarkupAnnotationData](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Collections.Generic;
namespace ModifyExistingMarkupAnnotation {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\..\\Document.pdf");
                // Get a list of annotations in the first document page.
                IList<PdfMarkupAnnotationData> annotations = processor.GetMarkupAnnotationData(1);
                if (annotations.Count > 0) {
                    // Change the properties of the first annotation.
                    annotations[0].Author = "Bill Smith";
                    annotations[0].Contents = "Important!";
                    annotations[0].Color = new PdfRGBColor(0.6, 0.7, 0.3);
                    annotations[0].Opacity = 0.2;
                    // Save the result document.
                    processor.SaveDocument("..\\..\\..\\Result.pdf");
                }
                else
                    Console.WriteLine("The specified document page does not contain markup annotations!");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Collections.Generic
Namespace ModifyExistingMarkupAnnotation
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Get a list of annotations in the first document page.
                Dim annotations As IList(Of PdfMarkupAnnotationData) = processor.GetAnnotations(0)
                If annotations.Count > 0 Then
                    ' Change the properties of the first annotation.
                    annotations(0).Author = "Bill Smith"
                    annotations(0).Contents = "Important!"
                    annotations(0).Color = New PdfRGBColor(0.6, 0.7, 0.3)
                    annotations(0).Opacity = 0.2
                    ' Save the result document.
                    processor.SaveDocument("../..\Result.pdf")
                Else
                    Console.WriteLine("The specified document page does not contain any annotations.")
                End If
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

How to: Delete Particular Markup Annotations from a Page

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Text Markup Annotation](#) > [How to: Delete Particular Markup Annotations from a Page](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T549297>.

This example shows how to delete text markup annotations created by a particular author.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System.Linq;
namespace RemoveSpecificMarkupAnnotations {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../..\..\Document.pdf");
                for (int i = 0; i <= processor.Document.Pages.Count; i++) {
                    // Remove Bill Smith's markup annotations from a page.
                    processor.DeleteMarkupAnnotations(processor.GetMarkupAnnotationData(i)
                        .Where(annotation => annotation.Author.Contains("Bill Smith")));
                    // Save the result document.
                    processor.SaveDocument("../..\..\Result.pdf");
                }
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Linq
Namespace RemoveSpecificMarkupAnnotations
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\..\Document.pdf")
                For i As Integer = 0 To processor.Document.Pages.Count
                    ' Remove Bill Smith's markup annotations from a page.
                    processor.DeleteMarkupAnnotations(processor.GetMarkupA
                    ' Save the result document.
                    processor.SaveDocument("../..\..\Result.pdf")
                Next i
            End Using
        End Sub
    End Class
End Namespace
```

See Also[Getting Started](#)

Extract Content from a PDF Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Extract Content from a PDF Document](#)

This section contains the following examples:

- [How to: Extract Images from a Document](#)
- [How to: Extract Text from a Document](#)
- [How to: Search Text in a Document](#)

How to: Extract Images from a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Extract Content from a PDF Document](#) > [How to: Extract Images from a Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T161599>.

This example illustrates the use of the [PdfDocumentProcessor.GetImages](#) method for obtaining document bitmaps in code by using [PDF Document API](#).

C#

```
(Program.cs)
using System;
using System.Collections.Generic;
using System.Drawing;
using DevExpress.Pdf;
// ...

static void Main(string[] args) {
    PdfDocumentProcessor processor = new PdfDocumentProcessor();
    processor.LoadDocument(@"..\..\Demo.pdf");
    int xCount = 8;
    int yCount = 2;
    double cardWidth = 150.5; // Measured in points (equals 2.09 inches).
    double cardHeight = 442; // Measured in points (equals 6.138 inches).
    double xMargin = 122; // Measured in points (equals 1.694 inches).
    double yMargin = 77; // Measured in points (equals 1.069 inches).
    double yCoord = yMargin;
    for (int y = 0; y < yCount; y++, yCoord += cardHeight) {
        double xCoord = xMargin;
        for (int x = 0; x < xCount; x++, xCoord += cardWidth) {
            PdfDocumentArea area = new PdfDocumentArea(1,
                new PdfRectangle(xCoord, yCoord, xCoord + cardWidth, yCoord + cardHeight));
            IList<Bitmap> bitmaps = processor.GetImages(area);
            if (bitmaps.Count != 0) {
                bitmaps[0].Save(String.Format(@"{0}_{1}.bmp", x, y));
                bitmaps[0].Dispose();
            }
            Console.WriteLine(bitmaps.Count.ToString());
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports System
Imports System.Collections.Generic
Imports System.Drawing
Imports DevExpress.Pdf
' ...

Shared Sub Main(ByVal args() As String)
    Dim processor As New PdfDocumentProcessor()
    processor.LoadDocument("../\\...\\Demo.pdf")
    Dim xCount As Integer = 8
    Dim yCount As Integer = 2
    Dim cardWidth As Double = 150.5 ' Measured in points (equals 2
    Dim cardHeight As Double = 442 ' Measured in points (equals 6.
    Dim xMargin As Double = 122 ' Measured in points (equals 1.694
    Dim yMargin As Double = 77 ' Measured in points (equals 1.069
    Dim yCoord As Double = yMargin
    Dim y As Integer = 0
    Do While y < yCount
        Dim xCoord As Double = xMargin
        Dim x As Integer = 0
        Do While x < xCount
            Dim area As New PdfDocumentArea(1, New PdfRectangle(xC
            Dim bitmaps As IList(Of Bitmap) = processor.GetImages(
            If bitmaps.Count <> 0 Then
                bitmaps(0).Save(String.Format("{0}_{1}.bmp", x, y)
                bitmaps(0).Dispose()
            End If
            Console.WriteLine(bitmaps.Count.ToString())
            x += 1
            xCoord += cardWidth
        Loop
        y += 1
        yCoord += cardHeight
    Loop
End Sub
```

See Also

[How to: Extract Text from a Document](#)

How to: Extract Text from a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Extract Content from a PDF Document](#) > [How to: Extract Text from a Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E5025>.

This tutorial describes how to extract the text of a PDF file at runtime using the [PDF Document API](#).

To extract the text of a PDF file, do the following.

1. Create a [PdfDocumentProcessor](#).
2. To open a PDF file, pass a stream that contains the document data to the [PdfDocumentProcessor.LoadDocument](#) method.
3. After the document is loaded, you can extract its plain text using the [PdfDocumentProcessor.Text](#) property.

The following code implements this functionality.

C#

```
(MainForm.cs)
string ExtractTextFromPDF(string filePath) {
    string documentText = "";
    try {
        using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
            documentProcessor.LoadDocument(filePath);
            documentText = documentProcessor.Text;
        }
    }
    catch { }
    return documentText;
}
```

Visual Basic

```
(MainForm.vb)
Private Function ExtractTextFromPDF(ByVal filePath As String) As String
    Dim documentText As String = ""
    Try
        Using documentProcessor As New PdfDocumentProcessor()
            documentProcessor.LoadDocument(filePath)
            documentText = documentProcessor.Text
        End Using
    Catch
    End Try
    Return documentText
End Function
```

See Also

[How to: Extract Images from a Document](#)

How to: Search Text in a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Extract Content from a PDF Document](#) > [How to: Search Text in a Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E5025>.

This example demonstrates how to count the occurrences of words in a document text.

To accomplish this task, do the following.

- Create a [PdfDocumentProcessor](#).
 - To load a PDF file, pass a file path to the [PdfDocumentProcessor.LoadDocument](#) method.
 - Call the [PdfDocumentProcessor.FindText](#) method with a string parameter containing the text to search in the document.
- After the method returns the results of the search operation, repeat the call until each occurrence of the specified text is found in the document.

Note

Alternatively, use an overloaded [PdfDocumentProcessor.FindText](#) method to specify additional search parameters, such as case sensitivity and search direction.

To navigate through the text of the document, use the [PdfDocumentProcessor.PrevWord](#) and [PdfDocumentProcessor.NextWord](#) methods that return a **PdfWord** object containing a word located at the current search position.

C#

```
(MainForm.cs)
using DevExpress.Pdf;
using DevExpress.XtraBars;
using DevExpress.XtraEditors;
using System.IO;
using System.Text;
using System.Windows.Forms;
// ...

private int WordCount(string filePath, string word) {
    int count = 0;
    try {
        using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
            documentProcessor.LoadDocument(filePath);
            while (documentProcessor.FindText(word).Status == PdfTextSearchStatus.Found) {
                count++;
            }
        }
    }
    catch { }
    return count;
}
```

Visual Basic

```
(MainForm.vb)
Imports DevExpress.Pdf
Imports DevExpress.XtraBars
Imports DevExpress.XtraEditors
Imports System.IO
Imports System.Text
Imports System.Windows.Forms

' ...

Private Function WordCount(ByVal filePath As String, ByVal word As String) As Integer
    Dim count As Integer = 0
    Try
        Using documentProcessor As New PdfDocumentProcessor()
            documentProcessor.LoadDocument(filePath)
            Do While documentProcessor.FindText(word).Status = PdfFindTextStatus.Found
                count += 1
            Loop
        End Using
    Catch
    End Try
    Return count
End Function
```

Manage Pages of a PDF Document


[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Manage Pages of a PDF Document](#)

This section contains the following examples:


- [How to: Extract Pages from a Document](#)
- [How to: Delete Pages from a Document](#)
- [How to: Merge Documents into a Single PDF File](#)
- [How to: Rotate Pages](#)

How to: Extract Pages from a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Manage Pages of a PDF Document](#) > [How to: Extract Pages from a Document](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T374804>.

This example shows how to extract the first page from a document into a separate PDF document.

To accomplish this task:

- load a source document using the [PdfDocumentProcessor.LoadDocument](#) method;
 - create a target document with no pages using one of the [PdfDocumentProcessor.CreateEmptyDocument](#) overloaded methods;
 - add the first page from the source document to the target document.
- The target document can be accessed using the [PdfDocumentProcessor.Document](#) property. To access a collection of pages within the target document, use the PdfDocument.Pages property. Then, you're able to add pages to the collection using the **Add** method. To access the first page in the page collection of a source document, use the zero page index.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace ExtractFirstPage {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor source = new PdfDocumentProcessor()) {
                source.LoadDocument("..\\..\\..\\Document.pdf");
                using (PdfDocumentProcessor target = new PdfDocumentProcessor()) {
                    target.CreateEmptyDocument("..\\..\\..\\ExtractedFirstPage.pdf");
                    target.Document.Pages.Add(source.Document.Pages[0]);
                }
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace ExtractFirstPage
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using source As New PdfDocumentProcessor()
                source.LoadDocument("../..\\Document.pdf")
                Using target As New PdfDocumentProcessor()
                    target.CreateEmptyDocument("../..\\ExtractedFirstPage.p
                    target.Document.Pages.Add(source.Document.Pages(0))
                End Using
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

How to: Delete Pages from a Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Manage Pages of a PDF Document](#) > [How to: Delete Pages from a Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T114310>.

This example illustrates how to use the PDF Document API component for deleting pages from a document.

In this example, the [PdfDocumentProcessor.DeletePage](#) method is called 55 times to delete each odd numbered page in the **TextDelete** document (contains 109 pages), starting from the last odd numbered page.

The result is saved to the **Deleted** document, which contains only even pages, by calling the [PdfDocumentProcessor.SaveDocument](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
// ...
class Program {
    static void Main(string[] args) {
        using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) {
            pdfDocumentProcessor.LoadDocument("..\\..\\docs\\TextDelete.pdf");
            for (int i = pdfDocumentProcessor.Document.Pages.Count / 2; i >= 0; i--)
                pdfDocumentProcessor.DeletePage(i * 2 + 1);
            pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Deleted.pdf");
        }
    }
}
```


Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
' ...
Friend Class Program
    Shared Sub Main(ByVal args() As String)
        Using pdfDocumentProcessor As New PdfDocumentProcessor()
            pdfDocumentProcessor.LoadDocument("..\\..\\docs\\TextDelete.p
            For i As Integer = pdfDocumentProcessor.Document.Pages.Cou
                pdfDocumentProcessor.DeletePage(i * 2 + 1)
            Next i
            pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Deleted.pdf"
        End Using
    End Sub
End Class
```


See Also
[Getting Started](#)

How to: Merge Documents into a Single PDF File

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Manage Pages of a PDF Document](#) > [How to: Merge Documents into a Single PDF File](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T114298>.

This example illustrates how to use the PDF Document API component for merging pages of two separate PDF files into a single PDF file.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace PdfMergeExample {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) {
                pdfDocumentProcessor.CreateEmptyDocument("..\\..\\docs\\Merged.pdf");
                pdfDocumentProcessor.AppendDocument("..\\..\\docs\\TextMerge1.pdf");
                pdfDocumentProcessor.AppendDocument("..\\..\\docs\\TextMerge2.pdf");
            }
        }
    }
}
```


Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace PdfMergeExample
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using pdfDocumentProcessor As New PdfDocumentProcessor()
                pdfDocumentProcessor.CreateEmptyDocument("..\\..\\docs\\Merge
                pdfDocumentProcessor.AppendDocument("..\\..\\docs\\TextMerge1
                pdfDocumentProcessor.AppendDocument("..\\..\\docs\\TextMerge2
            End Using
        End Sub
    End Class
End Namespace
```


See Also
[Getting Started](#)

How to: Rotate Pages

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Manage Pages of a PDF Document](#) > [How to: Rotate Pages](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T114305>.

This example illustrates how to use the PDF Document API component for rotating pages.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace PdfPageRotationExample {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) {
                pdfDocumentProcessor.LoadDocument("..\\..\\docs\\TextRotate.pdf");
                int angle = 0;
                foreach (PdfPage page in pdfDocumentProcessor.Document.Pages) {
                    angle = (angle + 90) % 360;
                    page.Rotate = angle;
                }
                pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Rotated.pdf");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace PdfPageRotationExample
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using pdfDocumentProcessor As New PdfDocumentProcessor()
                pdfDocumentProcessor.LoadDocument("..\\..\\docs\\TextRotate.p
                Dim angle As Integer = 0
                For Each page As PdfPage In pdfDocumentProcessor.Document.
                    angle = (angle + 90) Mod 360
                    page.Rotate = angle
                Next page
                pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Rotated.pdf"
            End Using
        End Sub
    End Class
End Namespace
```

See Also
[Getting Started](#)

Document Protection


[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Protection](#)

This section contains the following examples:


- [How to: Protect a PDF Document with a Password](#)
- [How to: Add a Digital Signature into a PDF Document](#)

How to: Protect a PDF Document with a Password

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Protection](#) > [How to: Protect a PDF Document with a Password](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T243764>.

This example shows how a PDF document can be protected using both the owner and user passwords.

For more information, see the [Protecting a Document](#) topic.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace PDFPasswordProtection {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) {
                // Load a PDF document.
                pdfDocumentProcessor.LoadDocument("..\\..\\Demo.pdf");
                // Specify printing, data extraction, modification, and interactivity permissions.
                PdfEncryptionOptions encryptionOptions = new PdfEncryptionOptions();
                encryptionOptions.PrintingPermissions = PdfDocumentPrintingPermissions.Allowed;
                encryptionOptions.DataExtractionPermissions = PdfDocumentDataExtractionPermissions.NotAllowed;
                encryptionOptions.ModificationPermissions = PdfDocumentModificationPermissions.DocumentAsReadOnly;
                encryptionOptions.InteractivityPermissions = PdfDocumentInteractivityPermissions.Allowed;
                // Specify the owner and user passwords for the document.
                encryptionOptions.OwnerPasswordString = "OwnerPassword";
                encryptionOptions.UserPasswordString = "UserPassword";
                // Specify the 256-bit AES encryption algorithm.
                encryptionOptions.Algorithm = PdfEncryptionAlgorithm.AES256;
                // Save the protected document with encryption settings.
                pdfDocumentProcessor.SaveDocument("..\\..\\ProtectedDocument.pdf", new PdfSaveOptions() {
                })
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace PDFPasswordProtection
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using pdfDocumentProcessor As New PdfDocumentProcessor()
                ' Load a PDF document.
                pdfDocumentProcessor.LoadDocument("../..\Demo.pdf")
                ' Specify printing, data extraction, modification, and int
                Dim encryptionOptions As New PdfEncryptionOptions()
                encryptionOptions.PrintingPermissions = PdfDocumentPrintin
                encryptionOptions.DataExtractionPermissions = PdfDocumentD
                encryptionOptions.ModificationPermissions = PdfDocumentMod
                encryptionOptions.InteractivityPermissions = PdfDocumentIn
                ' Specify the owner and user passwords for the document.
                encryptionOptions.OwnerPasswordString = "OwnerPassword"
                encryptionOptions.UserPasswordString = "UserPassword"
                ' Specify the 256-bit AES encryption algorithm.
                encryptionOptions.Algorithm = PdfEncryptionAlgorithm.AES25
                ' Save the protected document with encryption settings.
                pdfDocumentProcessor.SaveDocument("../..\ProtectedDocument
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

How to: Add a Digital Signature into a PDF Document

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Document Protection](#) > [How to: Add a Digital Signature into a PDF Document](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T243905>.

This example illustrates how to apply a digital signature to a PDF document.

To accomplish this task, do the following:

- Create a PDF Document API component represented by an instance of the [PdfDocumentProcessor](#) class.
- Load a document from a file using the [PdfDocumentProcessor.LoadDocument](#) method.
- Create a certificate using a certificate file name and a password to access the certificate.
- Create a document digital signature represented by a PdfSignature object using the certificate;
- Specify signing location, contact info and reason using the PdfSignature.Location, PdfSignature.ContactInfo and PdfSignature.Reason properties, respectively.
- Save the signed document with signing information by calling the [PdfDocumentProcessor.SaveDocument](#) method.

C#

(Program.cs)
using DevExpress.Pdf;
using System.Security.Cryptography.X509Certificates;
namespace PDFSignature {
 class Program {
 static void Main(string[] args) {
 // Create a PDF document processor.
 using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
 // Load a PDF document.
 documentProcessor.LoadDocument(@"..\..\Demo.pdf");
 // Load a certificate from a file.
 X509Certificate2 cert = new X509Certificate2(@"..\..\SignDemo.pfx", "dxdemo");
 // Create a PDF signature and specify signing location, contact info and reason.
 PdfSignature signature = new PdfSignature(cert) {
 Location = "Location",
 ContactInfo = "ContactInfo",
 Reason = "Reason"
 };
 // Save the signed document.
 documentProcessor.SaveDocument(@"..\..\SignedDocument.pdf", new PdfSaveOptions() { Signatu

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Security.Cryptography.X509Certificates
Namespace PDFSignature
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF document processor.
            Using documentProcessor As New PdfDocumentProcessor()
                ' Load a PDF document.
                documentProcessor.LoadDocument("..\\..\\Demo.pdf")
                ' Load a certificate from a file.
                Dim cert As New X509Certificate2("..\\..\\SignDemo.pfx", "dx")
                ' Create a PDF signature and specify signing location, content, and certificate.
                Dim signature As New DevExpress.Pdf.PdfSignature(cert) With {
                    .Location = PdfSignatureLocation.Content,
                    .Content = "Signed Document",
                    .Certificate = cert
                }
                ' Save the signed document.
                documentProcessor.SaveDocument("..\\..\\SignedDocument.pdf", signature)
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[Getting Started](#)

Printing


[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Printing](#)

This section contains the following examples:


- [How to: Use the PDF Printer Settings](#)
- [How to: Customize PDF Print Output](#)

How to: Use the PDF Printer Settings

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Printing](#) > [How to: Use the PDF Printer Settings](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T172274>.

This example shows how to print a document with custom printer settings.

C#

```
(Program.cs)
// Developer Express Code Central Example:
// How to use the PDF printer settings
using DevExpress.Pdf;
namespace PdfProcessorPrinterOptions {
    class Program {
        static void Main(string[] args) {
            // Create a Pdf Document Processor instance and load a PDF into it.
            PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor();
            documentProcessor.LoadDocument(@"..\..\Demo.pdf");
            // Declare the PDF printer settings.
            PdfPrinterSettings pdfPrinterSettings = new PdfPrinterSettings();
            // Specify the PDF printer settings.
            pdfPrinterSettings.PageOrientation = PdfPrintPageOrientation.Portrait;
            pdfPrinterSettings.PageNumbers = new int[] { 1, 3, 4, 5 };
            // Setting the PdfPrintScaleMode property to CustomScale requires
            // specifying the Scale property, as well.
            pdfPrinterSettings.ScaleMode = PdfPrintScaleMode.CustomScale;
            pdfPrinterSettings.Scale = 90;
            // Print the document using the specified printer settings.
            documentProcessor.Print(pdfPrinterSettings);
        }
    }
}
```

Visual Basic


```
(Program.vb)
' Developer Express Code Central Example:
' How to use the PDF printer settings
Imports DevExpress.Pdf
Namespace PdfProcessorPrinterOptions
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a Pdf Document Processor instance and load a PDF into
            Dim documentProcessor As New PdfDocumentProcessor()
            documentProcessor.LoadDocument("../..\Demo.pdf")
            ' Declare the PDF printer settings.
            Dim pdfPrinterSettings As New PdfPrinterSettings()
            ' Specify the PDF printer settings.
            pdfPrinterSettings.PageOrientation = PdfPrintPageOrientation.P
            pdfPrinterSettings.PageNumbers = New Integer() { 1, 3, 4, 5 }
            ' Setting the PdfPrintScaleMode property to CustomScale requir
            ' specifying the Scale property, as well.
            pdfPrinterSettings.ScaleMode = PdfPrintScaleMode.CustomScale
            pdfPrinterSettings.Scale = 90
            ' Print the document using the specified printer settings.
            documentProcessor.Print(pdfPrinterSettings)
        End Sub
    End Class
End Namespace
```

See Also


[Getting Started](#)

How to: Customize PDF Print Output

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [Printing](#) > [How to: Customize PDF Print Output](#)

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T334688>.

This example shows how to customize print output and change settings for a page to be printed.

A document is printed with the specified initial printer settings using the [PdfDocumentProcessor.Print](#) method. In this example, the page numbers that should be printed in a document are specified using the PdfPrinterSettings.PageNumbers property.

When a page is printed, page settings for this page are queried via the [PdfDocumentProcessor.QueryPageSettings](#) event.

In this example, the second page of a document is printed in landscape orientation by setting the **PdfQueryPageSettingsEventArgs.PageSettings.Landscape** property to **true** when the [PdfDocumentProcessor.QueryPageSettings](#) event is handled.

When the document is sent to a printer, the [PdfDocumentProcessor.PrintPage](#) event is raised.

To draw an additional image on a printed page, the **Graphics** property of the PdfPrintPageEventArgs is used when the [PdfDocumentProcessor.PrintPage](#) event is handled. This image is drawn on each page by calling the **Graphics.DrawImage** method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System.Drawing;
namespace CustomizePrintSettings {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor instance and load a PDF into it.
            using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
                documentProcessor.LoadDocument(@"..\..\Demo.pdf");
                // Declare the PDF printer settings.
                PdfPrinterSettings settings = new PdfPrinterSettings();
                // Specify the page numbers to be printed.
                settings.PageNumbers = new int[] { 1, 2, 3, 4, 5, 6 };
                // Handle the PrintPage event to specify print output.
                documentProcessor.PrintPage += OnPrintPage;
                // Handle the QueryPageSettings event to customize settings for a page to be printed.
                documentProcessor.QueryPageSettings += OnQueryPageSettings;
                // Print the document using the specified printer settings.
                documentProcessor.Print(settings);
                // Unsubscribe from PrintPage and QueryPageSettings events.
                documentProcessor.PrintPage -= OnPrintPage;
                documentProcessor.QueryPageSettings -= OnQueryPageSettings;
            }
        }
        private static void OnQueryPageSettings(object sender, PdfQueryPageSettingsEventArgs e) {
            // Print the second page in landscape size.
            if (e.PageNumber == 2) {
                e.PageSettings.Landscape = true;
            }
            else e.PageSettings.Landscape = false;
        }
        // Specify what happens when the PrintPage event is raised.
        private static void OnPrintPage(object sender, PdfPrintPageEventArgs e) {
            // Draw a picture on each printed page.
            using (Bitmap image = new Bitmap(@"..\..\DevExpress.png"))
                e.Graphics.DrawImage(image, new RectangleF(10, 30, image.Width / 2, image.Height / 2));
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Drawing
Namespace CustomizePrintSettings
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor instance and load a PDF into
            Using documentProcessor As New PdfDocumentProcessor()
                documentProcessor.LoadDocument("../..\Demo.pdf")
                ' Declare the PDF printer settings.
                Dim settings As New PdfPrinterSettings()
                ' Specify the page numbers to be printed.
                settings.PageNumbers = New Integer() { 1, 2, 3, 4, 5, 6 }
                ' Handle the PrintPage event to specify print output.
                AddHandler documentProcessor.PrintPage, AddressOf OnPrintPage
                ' Handle the QueryPageSettings event to customize settings
                AddHandler documentProcessor.QueryPageSettings, AddressOf OnQueryPageSettings
                ' Print the document using the specified printer settings.
                documentProcessor.Print(settings)
                ' Unsubscribe from PrintPage and QueryPageSettings events.
                RemoveHandler documentProcessor.PrintPage, AddressOf OnPrintPage
                RemoveHandler documentProcessor.QueryPageSettings, AddressOf OnQueryPageSettings
            End Using
        End Sub
        Private Shared Sub OnQueryPageSettings(ByVal sender As Object, ByVal e As PdfQueryPageSettings)
            ' Print the second page in landscape size.
            If e.PageNumber = 2 Then
                e.PageSettings.Landscape = True
            Else
                e.PageSettings.Landscape = False
            End If
        End Sub
        ' Specify what happens when the PrintPage event is raised.
        Private Shared Sub OnPrintPage(ByVal sender As Object, ByVal e As PdfPrintPageEventArgs)
            ' Draw a picture on each printed page.
            Using image As New Bitmap("../..\DevExpress.png")
                e.Graphics.DrawImage(image, New RectangleF(10, 30, image.Width, image.Height))
            End Using
        End Sub
    End Class
End Namespace
```

See Also
[Getting Started](#)

Export a PDF Document to an Image

How to: Export a PDF Document to a Bitmap

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [How to: Export a PDF Document to a Bitmap](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

The following example demonstrates how to export pages to bitmap images.

Follow the steps below to perform the export:

- Create a [PdfDocumentProcessor](#) instance and load the PDF document using the overloaded [PdfDocumentProcessor.LoadDocument](#) method.
- Call the [PdfDocumentProcessor.CreateBitmap](#) method using the page number and the **largestEdgeLength** parameter (measured in pixels). This parameter determines the output image's height for pages in the portrait orientation and width for landscape pages.
- Save the converted bitmap images to a file using the overloaded **Bitmap.Save** method.

Show Me

The complete sample project is available at <https://github.com/DevExpress-Examples/how-to-export-a-pdf-document-to-multi-page-tiff-and-bitmap>

C#

```
using DevExpress.Pdf;
using System.Drawing;
namespace ExportToBitmap {
    class Program {
        static void Main(string[] args) {
            int largestEdgeLength = 1000;
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../\\..\\Document.pdf");
                for (int i = 1; i <= processor.Document.Pages.Count; i++) {
                    // Export pages to bitmaps.
                    Bitmap image = processor.CreateBitmap(i, largestEdgeLength);
                    // Save the bitmaps.
                    image.Save("../\\..\\MyBitmap" + i + ".bmp");
                }
            }
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
Imports System.Drawing
Namespace ExportToBitmap
    Class Program
        Private Shared Sub Main(ByVal args As String())
            Dim largestEdgeLength As Integer = 1000
            Using processor As PdfDocumentProcessor = New PdfDocumentProcessor()
                processor.LoadDocument("../..\Document.pdf")
                For i As Integer = 1 To processor.Document.Pages.Count
                    Dim image As Bitmap = processor.CreateBitmap(i, largestEdgeLength)
                    image.Save("../..\MyBitmap" & i & ".bmp")
                Next
            End Using
        End Sub
    End Class
End Namespace
```

How to: Export a PDF Document to a Multi-Page Tiff

[Office File API](#) > [PDF Document API](#) > [Examples](#) > [How to: Export a PDF Document to a Multi-Page Tiff](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

The following example shows how to export pages to a multi-page Tiff image.

Follow the steps below to perform the export:

- Create a [PdfDocumentProcessor](#) instance and load the required PDF document using the overloaded [PdfDocumentProcessor.LoadDocument](#) method.
- Call one of the [PdfDocumentProcessor.CreateTiff](#) overloaded methods using, for example, the file path where the generated image should be located, the **largestEdgeLength** parameter measured in pixels and page numbers. The **largestEdgeLength** parameter determines images' height for pages in the portrait orientation and width for landscape pages.

Show Me

The complete sample project is available at <https://github.com/DevExpress-Examples/how-to-export-a-pdf-document-to-multi-page-tiff-and-bitmap>

C#

```
using DevExpress.Pdf;
namespace ExportToTiff {
    class Program {
        static void Main(string[] args) {
            int largestEdgeLength = 1000;
            int[] pageNumbers = new int[] { 1, 3, 5 };
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\Document.pdf");
                // Export pages to a multi-page tiff image.
                processor.CreateTiff("..\\..\\Image.tiff", largestEdgeLength, pageNumbers);
            }
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
Namespace ExportToTiff
    Class Program
        Private Shared Sub Main(ByVal args As String())
            Dim largestEdgeLength As Integer = 1000
            Dim pageNumbers As Integer() = New Integer() {1, 3, 5}
            Using processor As PdfDocumentProcessor = New PdfDocumentProcessor()
                processor.LoadDocument("..\\..\\Document.pdf")
                processor.CreateTiff("..\\..\\Image.tiff", largestEdgeLength, pageNumbers)
            End Using
        End Sub
    End Class
End Namespace
```

Excel Export Library

[Office File API](#) > [Excel Export Library](#)

The **Excel Export** library is designed to quickly generate a spreadsheet document in code and export it to [Excel formats](#). This library is developed to significantly improve performance and reduce memory consumption while generating spreadsheets. To maintain high performance standards, it does not create an internal document model, and writes data directly to a memory or file stream.

Unlike the multi-purpose [Spreadsheet Document API](#), which can load and edit spreadsheet documents, the **Excel Export** library is designed only for generating spreadsheet files in code, so that you can use the **Excel Export** API to export data from your application to Excel formats in the most efficient manner and with minimal memory usage (that is important to those of you who need to generate numerous Excel documents on the server side).

To learn more about the **Excel Export** library capabilities and restrictions, refer to the [Overview](#) section.

Main Features

The most significant features of the **Excel Export** library are listed below.

- Create basic elements of a spreadsheet document: worksheets, columns, rows and cells.
- Specify cell formatting: background color, alignment, borders, font settings, and number format.
- Apply different fonts to specific text regions within a cell.
- Add formulas and hyperlinks to a cell.
- Create merged cells.
- Add conditional formatting to worksheet cells.
- Freeze worksheet columns and rows.
- Display a worksheet from right to left.
- Insert pictures and add hyperlinks to them.
- Use filtering and grouping functionality.
- Create data validation criteria.
- Insert and modify sparklines.
- Define print options: add headers and footers to a worksheet printout, set a print area and specify print titles.
- Create and adjust tables.

Supported Formats

The **Excel Export** library supports the following file formats for data export.

- **XLSX**
Microsoft Office Open XML format - the default file format of Microsoft Excel, starting with Microsoft Excel 2007.
- **XLS**
Microsoft Excel 97-2003 binary file format.
- **CSV**
Comma Separated Values - the plain text format that uses comma characters as separators between cells.

See Also

- [Getting Started](#)
- [Examples](#)

Overview

[Office File API](#) > [Excel Export Library](#) > [Overview](#)

The topics in this section give you basic information on the architecture and functionality of the **XL Export** library and explain the differences between the **XL Export** library and [Spreadsheet Document API](#) to help you choose the right product for your next .NET project.

- [Excel Export and Spreadsheet Document API Architecture](#)
- [Excel Export and Spreadsheet Document API Feature Comparison](#)
- [Excel Export Specifications and Limits](#)
- [Product Structure](#)

Excel Export and Spreadsheet Document API Architecture

[Office File API](#) > [Excel Export Library](#) > [Overview](#) > [Excel Export and Spreadsheet Document API Architecture](#)

This document describes the key difference between the basic algorithms that [Spreadsheet Document API](#) and the [Excel Export Library](#) use for document generation.

Spreadsheet Document API

The [Spreadsheet Document API](#) architecture is based on its internal document model which stores all spreadsheet data in memory. This means that when you load a document, the Spreadsheet Document API component reads the file (using the appropriate file format, for example *XLSX*) and imports data (cell values, formulas, format settings, etc.) to the document model. At this point, you can use the spreadsheet API to modify the model or perform calculations over cell values. Once complete, you can call the **SaveDocument** or **ExportTo*** method and export the document model with modified data to the desired file format.

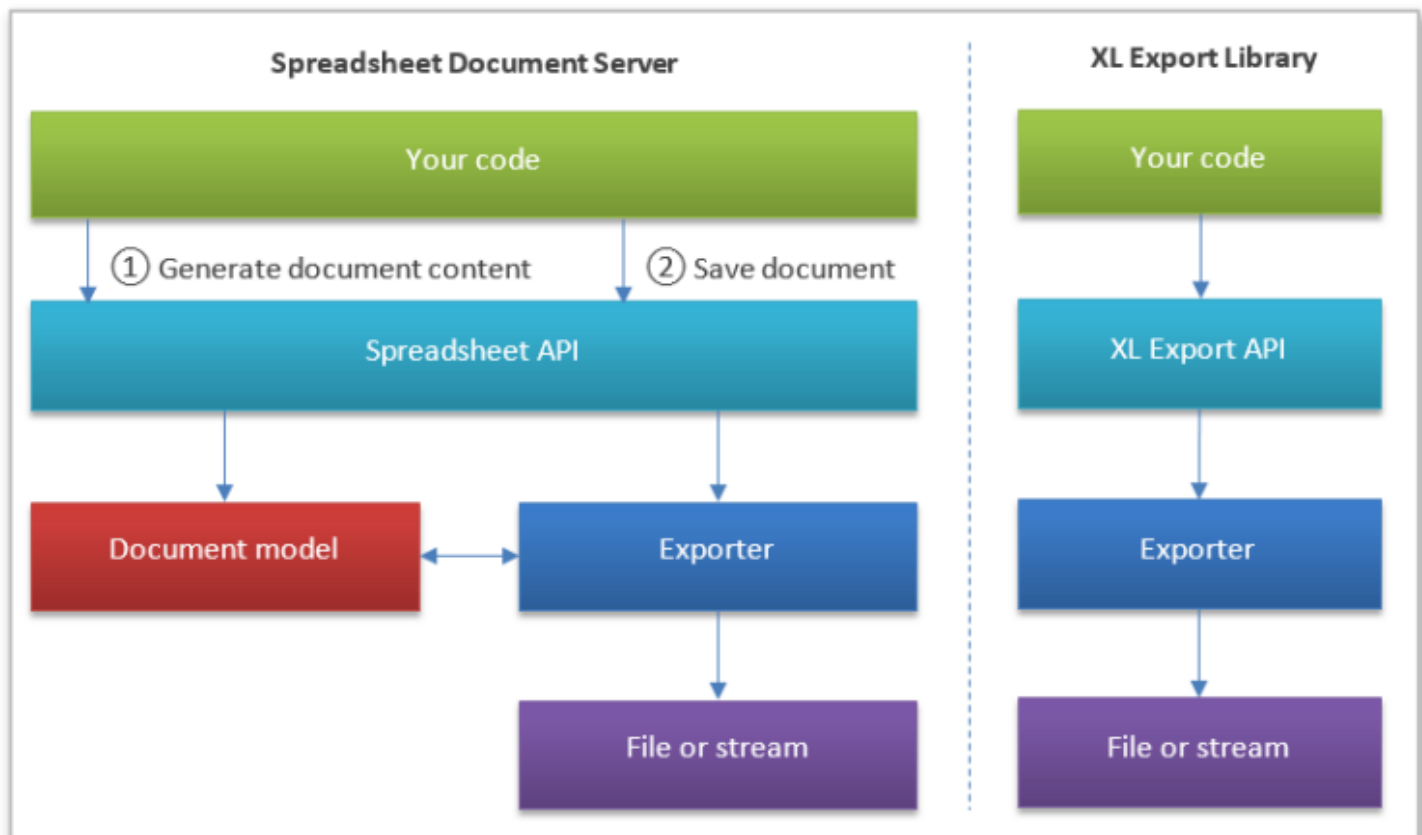
Generating a new document works in the same way. When you create a new workbook using API methods, the spreadsheet component creates a new document model, generates its content and then exports it to the specified file format. This universal approach allows you to use a single library to both read/edit and create new documents, but it can cause significant problems with performance: generation of very large Excel files will often lead to high memory consumption and associated time delays.

Excel Export Library

Unlike the **Spreadsheet Document API**, the [Excel Export Library](#) does not use an internal document model and writes data directly to the output stream in consecutive order: row by row, cell by cell. This approach allows it to generate Excel files at very high speeds and with very low memory consumption (because there is no need to put data into an in-memory document model and then export the model's content to the output stream).

So if you need an extremely fast and low-memory footprint engine to export data from your application into Excel formats, then the **Excel Export Library** is your best option. It is optimized for performance and allows you to create extremely large spreadsheet documents in the most efficient manner (that is very important when you have, for example, a website that needs to generate many Excel files).

The graphic below sums up the described algorithms used by the **Spreadsheet Document API** and **Excel Export Library** to generate spreadsheet documents.



See Also

[Excel Export and Spreadsheet Document API Feature Comparison](#)

Excel Export and Spreadsheet Document API Feature Comparison

[Office File API](#) > [Excel Export Library](#) > [Overview](#) > [Excel Export and Spreadsheet Document API Feature Comparison](#)

This topic provides information on feature differences between the [Excel Export Library](#) and [Spreadsheet Document API](#), so you can select the product that fully meets your demands.

- [Supported Document Formats](#)
- [Document Generation Capabilities](#)

Supported Document Formats

Document Format	Spreadsheet Document API	Excel Export Library
XLSX	✓	✓
XLS	✓	✓
CSV	✓	✓
XLSM	✓	
XLTX/XLTM/XLT	✓	
TXT (Tab Delimited Text)	✓	

Document Generation Capabilities

Feature	Spreadsheet Document API	Excel Export Library
Workbook/Worksheet	✓	✓
Rows/Columns	✓	✓
Cells	✓	✓
Merged cells	✓	✓
Formulas	✓	✓
Calculation engine	✓	
Custom functions	✓	
Defined names	✓	
External links	✓	
Cell styles	✓	
Cell formatting	✓	✓
Conditional formatting	✓	✓
.NET number formats		✓
Theme colors		✓
Pictures	✓	✓

Charts	✓	
Comments	✓	
Sparklines	✓	✓
Hyperlinks	✓	✓
Tables	✓	✓
Pivot Tables	✓	
Filtering	✓	✓
Grouping/Outline	✓	✓
Data validation	✓	✓
Frozen panes	✓	✓
Page breaks/ Page setup	✓	✓
Print area/options/titles	✓	✓
Headers/footers	✓	✓
Error checking options	✓	✓*
Protection	✓	
Encryption	✓	✓
Document properties	✓	✓

* Available for the entire worksheet data range, not a specific cell range.

Excel Export Specifications and Limits

[Office File API](#) > [Excel Export Library](#) > [Overview](#) > [Excel Export Specifications and Limits](#)

The **Excel Export Library** provides unified classes and interfaces used for exporting data to the most popular spreadsheet formats (XLSX, XLS and CVS). But be aware that the content of the resulting document highly depends on the output file format. For example, XLSX format supports all the features provided by the **Excel Export Library**, while the CSV file format keeps only cell values by saving them as text and does not support formatting, pictures, hyperlinks and other important features. That is why you should always be sure that you use the appropriate file format to export your data to avoid loss of important information and format settings.

Tip

To verify whether the certain functionality is supported by the file format to which you export your document, use properties of the `IXIDocumentOptions` object accessible using the `IXIDocument.Options` property.

The table below lists feature and formatting differences between the XLSX, XLS and CSV file formats.

Feature	XLSX	XLS	CSV
Maximum number of rows	1,048,576	65,536	2,147,483,647 ¹
Maximum number of columns	16,384	256	2,147,483,647 ²
Multi-part workbooks (with several worksheets)	✓	✓	✗
Columns			
Width	✓	✓	✗
Formatting options	✓	✓	Takes into account only number format settings. ²
Visibility state	✓	✓	✗
Rows			
Width	✓	✓	✗
Formatting options	✓	✓	Takes into account only number format settings. ²
Visibility state	✓	✓	✗
Cell Values	✓	✓	Cell values are exported as text.
Cell formatting			
Font	✓	✓	✗
Fill	✓	✓	✗
Alignment	✓	✓	✗
Borders	✓	✓	✗
Number format	✓	✓	✓ ²

Merged cells	✓	✓	✗
Formulas	✓	✓	✗
Conditional formatting	✓	✓ ³	✗
Pictures	✓	✓	✗
Sparklines	✓	✗	✗
Hyperlinks	✓	✓	✗
Filtering	✓	✓	✗
Grouping/Outline	✓	✓	✗
Data Validation	✓	✓	✗
Frozen panes	✓	✓	✗
Page breaks/Page setup	✓	✓	✗
Print area/options/titles	✓	✓	✗
Headers/Footers	✓	✓	✗
Encryption	✓	✓	✗
Error checking options	✓	✓	✗
Document properties	✓	✓	✗

¹ Note that Microsoft® Excel® and other spreadsheet applications allow you to import only 1,048,576 rows and 16,384 columns, so that you will be unable to view or edit your remaining data and this data may be lost if you try to save the modified workbook.

² The specified number format settings will be used to convert a cell value into a string during export when the **CsvDataAwareExporterOptions.UseCellNumberFormat** property is **true** (the default value). But note that when you open a CSV file in Microsoft® Excel® or any other spreadsheet application, each data column in the loaded file will be interpreted and formatted according to the default data format settings specified in the application.

³ Limited support: you cannot apply more than **three** conditional formats to a range of cells. Some conditional formatting types are not fully supported. For example, you cannot specify a data bar rule that uses a solid fill, border or bar direction settings, or contains negative bars. An icon set conditional formatting rule that uses an unsupported icon set arrangement (3 Triangles, 3 Stars, 5 Boxes, or custom icon set) will not be saved.

Product Structure

[Office File API](#) > [Excel Export Library](#) > [Product Structure](#)

Use the following links to access reference information about the most important classes and interfaces of the [Excel Export Library](#).

Class/Interface	Description
XIExport	A static class that allows you to create an exporter instance to perform data export to the specified Excel format (XLSX, XLS, or CSV). This is the entry point for programmatic generation of spreadsheet files with the Excel Export Library .
IXIExporter	An exporter used to create a spreadsheet document and export it to a stream.
IXIDocument	The main spreadsheet document or <i>workbook</i> .
IXISheet	An individual worksheet in a workbook.
IXIRow	An individual row in a worksheet.
IXIColumn	An individual column in a worksheet.
IXICell	A box at the intersection of a column and a row in a worksheet that can contain worksheet data, a formula and formatting.
XICellRange	A continuous range of cells in a worksheet.
IXIMergedCells	A collection of merged cells in a worksheet.
XIVariantValue	A data value contained in a cell.
XICellFormatting	Contains options to change cell format settings: fill, font, alignment, borders, and number format.
XIFill	Contains cell background attributes.
XIFont	Contains cell font attributes.
XICellAlignment	Contains alignment settings for a cell.
XIBorder	Provides access to the line characteristics of a cell border.
XINumberFormat	Allows you to specify a cell number format.
XIRichTextString	A rich formatted text in a cell.
XIConditionalFormatting	Allows you to create a conditional formatting rule(s) and apply it to the specified cell range(s). The following types of conditional formatting rules are available: <ul style="list-style-type: none"> • XICondFmtRuleAboveAverage - formats cells whose values are above or below the average in a range of cells; • XICondFmtRuleCellIs - formats cells whose values meet the criterion represented by a relational operator; • XICondFmtRuleBlanks - formats blank/non-blank cells; • XICondFmtRuleDuplicates - formats duplicate values; • XICondFmtRuleUnique - formats unique values; • XICondFmtRuleExpression - uses a formula to determine which cells to format;

	<ul style="list-style-type: none"> • <code>XICondFmtRuleTop10</code> - formats top/bottom ranked values; • <code>XICondFmtRuleSpecificText</code> - formats cells based on the text they contain; • <code>XICondFmtRuleTimePeriod</code> - formats dates that fall within a specified time period; • <code>XICondFmtRuleDataBar</code> - formats cells by using data bars; • <code>XICondFmtRuleIconSet</code> - formats cells by using icon sets; • <code>XICondFmtRuleColorScale</code> - formats cells using a gradation of two or three colors.
<code>DevExpress.Spreadsheet.XIFormulaParser</code>	Provides the capability to parse and validate string formulas. Using the <code>XIFormulaParser</code> in your code requires a reference to the DevExpress.Spreadsheet.v18.1.Core.dll assembly.
<code>IXIFormulaParameter</code>	Allows you to construct a formula from a combination of the most commonly used functions (static methods of the <code>XIFunc</code> class), arithmetic and relational operators (static methods of the <code>XIOper</code> class) and constants.
<code>XIExpression</code>	Allows you to create a tokenized representation of a formula.
<code>IXIFilterColumns</code>	A collection of columns in a worksheet to which filtering is applied.
<code>XIFilterColumn</code>	Identifies a particular column in the filtered range and contains filter information that has been applied to this column.
<code>IXIFilterCriteria</code>	<p>Allows you to specify the filter criteria. The following types of filters are supported:</p> <ul style="list-style-type: none"> • <code>XICustomFilters</code> - a custom filter that uses filter values and comparison operators to construct the filter expression; • <code>XIDynamicFilter</code> - a dynamic filter; • <code>XITop10Filter</code> - a "Top 10" filter that displays top/bottom ranked values; • <code>XIValuesFilter</code> - a filter by a list of cell values or date and time values; • <code>XIColorFilter</code> - a filter by cell color or font color.
<code>XIPageSetup</code>	Contains page settings: orientation, paper size, scaling options, printing DPI, etc.
<code>XIPrintOptions</code>	Contains printing options for a worksheet.
<code>XIPageMargins</code>	Allows you to specify page margins for a worksheet
<code>XIHeaderFooter</code>	Allows you to add headers and footers to a worksheet.
<code>XIPrintTitles</code>	Allows you to specify rows and columns that should be repeated on every printed page.
<code>XIDataValidation</code>	A data validation rule specified for a cell range(s) in a worksheet.
<code>IXIPicture</code>	An embedded image in a worksheet.
<code>XIHyperlink</code>	A hyperlink associated with a cell or cell range.
<code>XIPictureHyperlink</code>	A hyperlink associated with a picture.

IXITable	A table in a worksheet.
XISparklineGroup	A group of sparklines in a worksheet.
XISparkline	A single sparkline in the sparkline group.

See Also

[Getting Started](#)

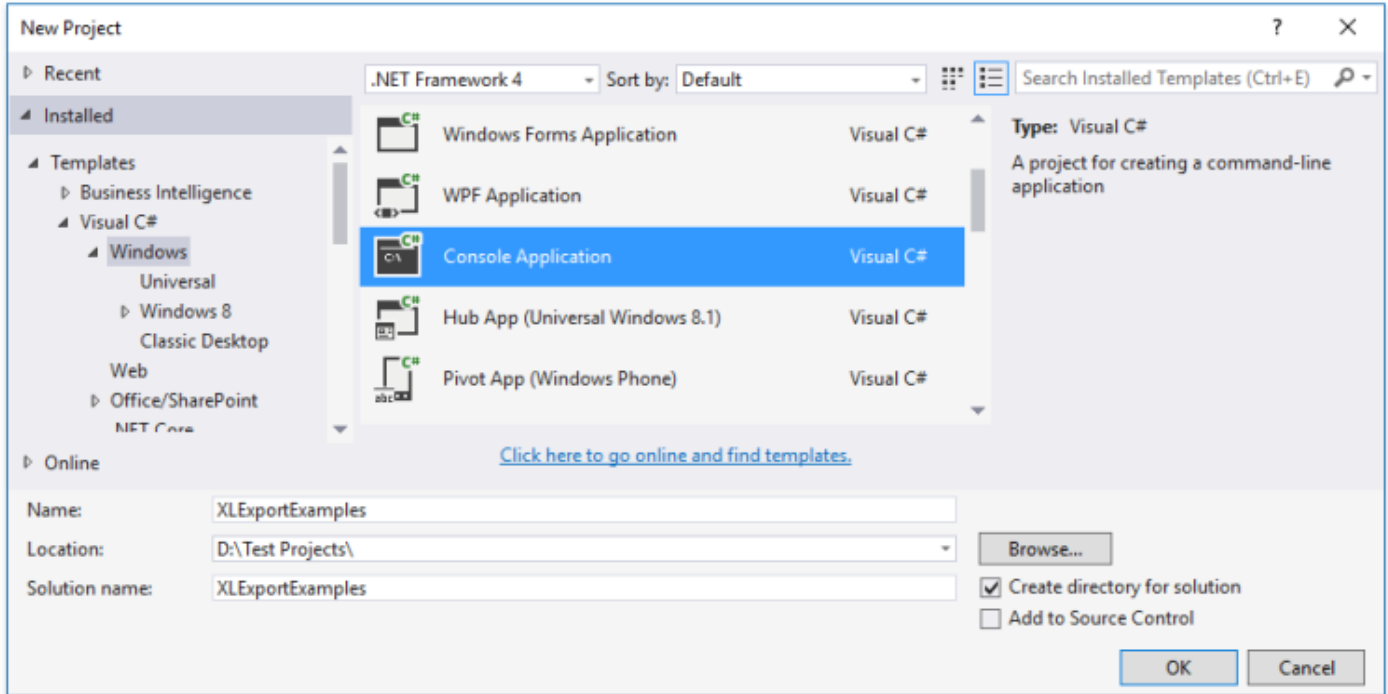
[Examples](#)

Getting Started

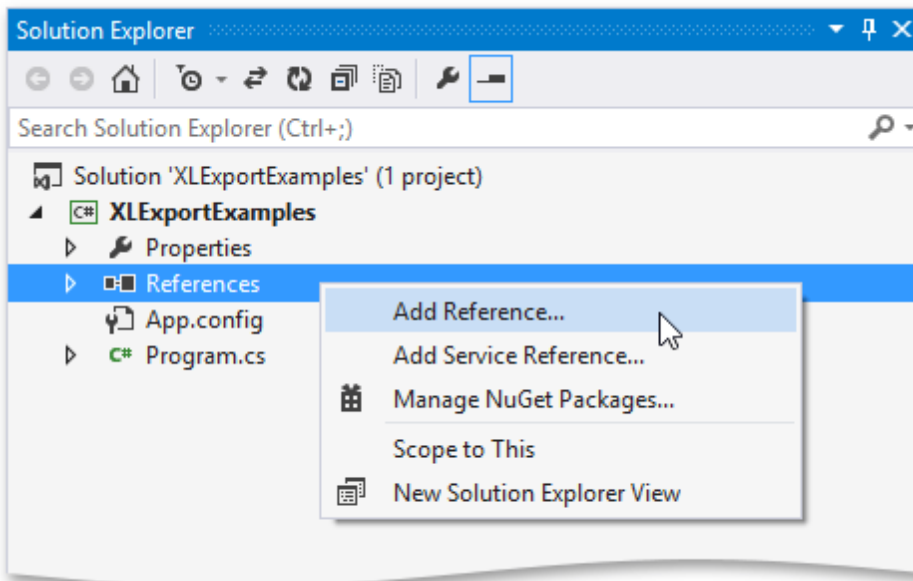
[Office File API](#) > [Excel Export Library](#) > [Getting Started](#)

To get started with the **Excel Export** library, perform the steps below. □

1. Start Visual Studio and create a new project by selecting **FILE | New | Project** in the main menu. In the invoked **New Project** dialog, select **Console Application**, specify the project name and location, and click **OK**.



2. In the **Solution Explorer**, right-click the **References** node and select **Add Reference...** in the context menu.



3. In the invoked **Reference Manager** dialog, add references to the following libraries.
 - DevExpress.Data.v18.1.dll
 - DevExpress.Printing.v18.1.Core.dll
4. Paste the code listed below in the **Main** method of the Program.cs file (**Main** procedure of the Module1.vb file for Visual Basic).

[Show Me](#)

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T242354>.

C#

```

(Program.cs)
using DevExpress.Export.Xl;
using System.IO;
// ...
namespace XlExportExamples
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create an exporter instance.
            XlExporter exporter = XlExport.CreateExporter(XlDocumentFormat.Xlsx);
            // Create the FileStream object with the specified file path.
            using (FileStream stream = new FileStream("Document.xlsx", FileMode.Create, FileAccess.ReadWrite))
            {
                // Create a new document and begin to write it to the specified stream.
                using (IXlDocument document = exporter.CreateDocument(stream)) {
                    // Add a new worksheet to the document.
                    using (IXlSheet sheet = document.CreateSheet()) {
                        // Specify the worksheet name.
                        sheet.Name = "Sales report";
                        // Create the first column and set its width.
                        using (IXlColumn column = sheet.CreateColumn()) {
                            column.WidthInPixels = 100;
                        }
                        // Create the second column and set its width.
                        using (IXlColumn column = sheet.CreateColumn()) {
                            column.WidthInPixels = 250;
                        }
                        // Create the third column and set the specific number format for its cells.
                        using (IXlColumn column = sheet.CreateColumn()) {
                            column.WidthInPixels = 100;
                            column.Formatting = new XlCellFormatting();
                            column.Formatting.NumberFormat = @"_([$-$-409]* #,##0.00_);_([$-$-409]* \(#,##0.00_);";
                        }
                        // Specify cell font attributes.
                        XlCellFormatting cellFormatting = new XlCellFormatting();
                        cellFormatting.Font = new XlFont();
                        cellFormatting.Font.Name = "Century Gothic";
                        cellFormatting.Font.SchemeStyle = XlFontSchemeStyles.None;
                        // Specify formatting settings for the header row.
                        XlCellFormatting headerRowFormatting = new XlCellFormatting();
                        headerRowFormatting.CopyFrom(cellFormatting);
                        headerRowFormatting.Font.Bold = true;
                        headerRowFormatting.Font.Color = XlColor.FromTheme(XlThemeColor.Light1, 0.0);
                        headerRowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent1, 0.0));
                        // Create the header row.
                        using (IXlRow row = sheet.CreateRow()) {
                            using (IXlCell cell = row.CreateCell()) {
                                cell.Value = "Region";
                                cell.ApplyFormatting(headerRowFormatting);
                            }
                            using (IXlCell cell = row.CreateCell()) {
                                cell.Value = "Product";
                                cell.ApplyFormatting(headerRowFormatting);
                            }
                            using (IXlCell cell = row.CreateCell()) {
                                cell.Value = "Sales";
                                cell.ApplyFormatting(headerRowFormatting);
                            }
                        }
                        // Generate data for the sales report.
                        string[] products = new string[] { "Camembert Pierrot", "Gorgonzola Telino", "Mascarpone", "Ricotta", "Tiramisu", "Zabaione", "Panna Cotta", "Creme Brulee" };
                        int[] amount = new int[] { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235 };
                        for (int i = 0; i < 8; i++)
                        {

```



```

(Program.vb)
Imports DevExpress.Export.Xl
Imports System.IO
' ...
Namespace XlExportExamples
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create an exporter instance.
            Dim exporter As IXlExporter = XlExport.CreateExporter(XlDocume
            ' Create the FileStream object with the specified file path.
            Using stream As New FileStream("Document.xlsx", FileMode.Creat
            ' Create a new document and begin to write it to the speci
            Using document As IXlDocument = exporter.CreateDocument(st
            ' Add a new worksheet to the document.
            Using sheet As IXlSheet = document.CreateSheet()
                ' Specify the worksheet name.
                sheet.Name = "Sales report"
                ' Create the first column and set its width.
                Using column As IXlColumn = sheet.CreateColumn()
                    column.WidthInPixels = 100
                End Using
                ' Create the second column and set its width.
                Using column As IXlColumn = sheet.CreateColumn()
                    column.WidthInPixels = 250
                End Using
                ' Create the third column and set the specific num
                Using column As IXlColumn = sheet.CreateColumn()
                    column.WidthInPixels = 100
                    column.Formatting = New XlCellFormatting()
                    column.Formatting.NumberFormat = "_([$$-409)*
                End Using
                ' Specify cell font attributes.
                Dim cellFormatting As New XlCellFormatting()
                cellFormatting.Font = New XlFont()
                cellFormatting.Font.Name = "Century Gothic"
                cellFormatting.Font.SchemeStyle = XlFontSchemeStyl
            ' Specify formatting settings for the header row.
            Dim headerRowFormatting As New XlCellFormatting()
            headerRowFormatting.CopyFrom(cellFormatting)
            headerRowFormatting.Font.Bold = True
            headerRowFormatting.Font.Color = XlColor.FromTheme
            headerRowFormatting.Fill = XlFill.SolidFill(XlColo
            ' Create the header row.
            Using row As IXlRow = sheet.CreateRow()
                Using cell As IXlCell = row.CreateCell()
                    cell.Value = "Region"
                    cell.ApplyFormatting(headerRowFormatting)
                End Using
                Using cell As IXlCell = row.CreateCell()
                    cell.Value = "Product"
                    cell.ApplyFormatting(headerRowFormatting)

```

```

        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Sales"
            cell.ApplyFormatting(headerRowFormatting)
        End Using
    End Using
    ' Generate data for the sales report.
    Dim products() As String = { "Camembert Pierrot",
    Dim amount() As Integer = { 6750, 4500, 3550, 4250
    For i As Integer = 0 To 7
        Using row As IXlRow = sheet.CreateRow()
            Using cell As IXlCell = row.CreateCell()
                cell.Value = If(i < 4, "East", "West")
                cell.ApplyFormatting(cellFormatting)
            End Using
            Using cell As IXlCell = row.CreateCell()
                cell.Value = products(i Mod 4)
                cell.ApplyFormatting(cellFormatting)
            End Using
            Using cell As IXlCell = row.CreateCell()
                cell.Value = amount(i)
                cell.ApplyFormatting(cellFormatting)
            End Using
        End Using
    Next i
    ' Enable AutoFilter for the created cell range.
    sheet.AutoFilterRange = sheet.DataRange
    ' Specify formatting settings for the total row.
    Dim totalRowFormatting As New XlCellFormatting()
    totalRowFormatting.CopyFrom(cellFormatting)
    totalRowFormatting.Font.Bold = True
    totalRowFormatting.Fill = XlFill.SolidFill(XlColor
    ' Create the total row.
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.ApplyFormatting(totalRowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Total amount"
            cell.ApplyFormatting(totalRowFormatting)
            cell.ApplyFormatting(XlCellAlignment.FromH
        End Using
        Using cell As IXlCell = row.CreateCell()
            ' Add values in the cell range C2 through
            cell.SetFormula(XlFunc.Subtotal(XlCellRang
            cell.ApplyFormatting(totalRowFormatting)
        End Using
    End Using
End Using
End Using
End Using

```

```
' Open the XLSX document using the default application.
System.Diagnostics.Process.Start("Document.xlsx")
End Sub
End Class
End Namespace
```

Run the project.

The following image shows the XLSX file generated after executing the code above (the document is opened in Microsoft® Excel®).

	A	B	C	D
1	Region	Product	Sales	
2	East	Camembert Pierrot	\$ 6,750.00	
3	East	Gorgonzola Telino	\$ 4,500.00	
4	East	Mascarpone Fabioli	\$ 3,550.00	
5	East	Mozzarella di Giovanni	\$ 4,250.00	
6	West	Camembert Pierrot	\$ 5,500.00	
7	West	Gorgonzola Telino	\$ 6,250.00	
8	West	Mascarpone Fabioli	\$ 5,325.00	
9	West	Mozzarella di Giovanni	\$ 4,235.00	
10	Total amount		\$ 40,360.00	
11				

See Also
[Examples](#)

Examples

[Office File API](#) > [Excel Export Library](#) > [Examples](#)

This section provides a full list of examples (grouped by features) contained in this help.

Workbooks

- [How to: Create a New Document](#)
- [How to: Password Protect a Workbook](#)
- [How to: Specify Document Properties for a Workbook](#)

Worksheets

- [How to: Create a New Worksheet](#)
- [How to: Set a Worksheet Name](#)
- [How to: Hide a Worksheet](#)
- [How to: Show and Hide Row and Column Headers](#)
- [How to: Show and Hide Gridlines](#)

Rows and Columns

- [How to: Create a Column](#)
- [How to: Create a Row](#)
- [How to: Hide a Row or Column](#)
- [How to: Specify Row Height and Column Width](#)
- [How to: Freeze Rows and Columns](#)

Cells

- [How to: Create a Worksheet Cell and Set Its Value](#)
- [How to: Merge Cells or Split Merged Cells](#)
- [How to: Add a Hyperlink to a Cell](#)

Formulas

- [How to: Create a Cell Formula](#)
- [How to: Create Shared Formulas](#)
- [How to: Create Subtotals](#)

Formatting

- [How to: Format a Cell](#)
- [How to: Apply Predefined Formatting to a Cell](#)
- [How to: Apply Themed Formatting to a Cell](#)
- [How to: Change Cell Background Color](#)
- [How to: Configure Cell Font Settings](#)
- [How to: Add Cell Borders](#)
- [How to: Align Cell Content](#)
- [How to: Specify Number Format for Cell Content](#)
- [How to: Apply Rich Formatting to the Cell Text](#)

Conditional Formatting

- [How to: Format Cell Values that are Above or Below the Average](#)
- [How to: Format Cells that are Less than, Greater than or Between a Value](#)
- [How to: Format Blank Cells](#)
- [How to: Format Unique or Duplicate Values](#)
- [How to: Use a Formula to Determine what Cells to Format](#)

- [How to: Format Top or Bottom Ranked Values](#)
- [How to: Format Cells based on the Text in the Cell](#)
- [How to: Format Cells with Dates](#)
- [How to: Format Cells Using Data Bars](#)
- [How to: Format Cells Using Icon Sets](#)
- [How to: Format Cells Using Color Scales](#)

Data Validation

- [How to: Apply Data Validation](#)

Tables

- [How to: Create a Table](#)
- [How to: Apply a Table Style](#)
- [How to: Apply Custom Formatting to a Table](#)
- [How to: Create a Calculated Column](#)

Filtering

- [Filtering](#)

Printing

- [How to: Specify Print Settings](#)
- [How to: Set Page Margins](#)
- [How to: Add Headers and Footers to a Worksheet Printout](#)
- [How to: Insert Page Breaks in a Worksheet](#)
- [How to: Print Titles on a Worksheet](#)

Sparklines

- [How to: Create Sparklines](#)
- [How to: Customize the Sparkline Appearance](#)
- [How to: Specify Sparkline Axis Settings](#)

Pictures

- [How to: Insert and Position a Picture in a Worksheet](#)
- [How to: Add a Hyperlink to a Picture](#)

See Also

[Getting Started](#)

Workbooks

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Workbooks](#)

This section contains the following examples:

- [How to: Create a New Document](#)
- [How to: Password Protect a Workbook](#)
- [How to: Specify Document Properties for a Workbook](#)

How to: Create a New Document

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Workbooks](#) > [How to: Create a New Document](#)

To create a new document using the Excel Export API, do the following.

1. Use the `XlExport.CreateExporter` method to create an object exposing the `IXIExporter` interface to perform an export to the specified file format.
2. Call the `IXIExporter.CreateDocument` method to create a new document and to write it to the specified file stream.

C#

```
// Create an exporter instance.
IXIExporter exporter = XlExport.CreateExporter(XlDocumentFormat.Xlsx);
// Create the FileStream object with the specified file path.
using (FileStream stream = new FileStream("Document.xlsx", FileMode.Create, FileAccess.ReadWrite))
{
    // Create a new document and write it to the specified stream.
    using (IXIDocument document = exporter.CreateDocument(stream))
    {
        // Specify the document culture.
        document.Options.Culture = CultureInfo.CurrentCulture;
    }
}
```

Visual Basic

```
' Create an exporter instance.
Dim exporter As IXIExporter = XlExport.CreateExporter(XlDocumentFormat.Xlsx)
' Create the FileStream object with the specified file path.
Using stream As New FileStream("Document.xlsx", FileMode.Create, FileAccess.ReadWrite)
    ' Create a new document and write it to the specified stream.
    Using document As IXIDocument = exporter.CreateDocument(stream)
        ' Specify the document culture.
        document.Options.Culture = CultureInfo.CurrentCulture
    End Using
End Using
```

After a workbook is generated, you can create other spreadsheet elements, such as [worksheets](#), [rows](#) and [columns](#), and [cells](#).

See Also

[How to: Create a New Worksheet](#)

How to: Password Protect a Workbook

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Workbooks](#) > [How to: Password Protect a Workbook](#)

The example below demonstrates how to encrypt a workbook and set a password to open it. To specify encryption options, create an instance of the EncryptionOptions class and pass it as a parameter to the IXIExporter.CreateDocument method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

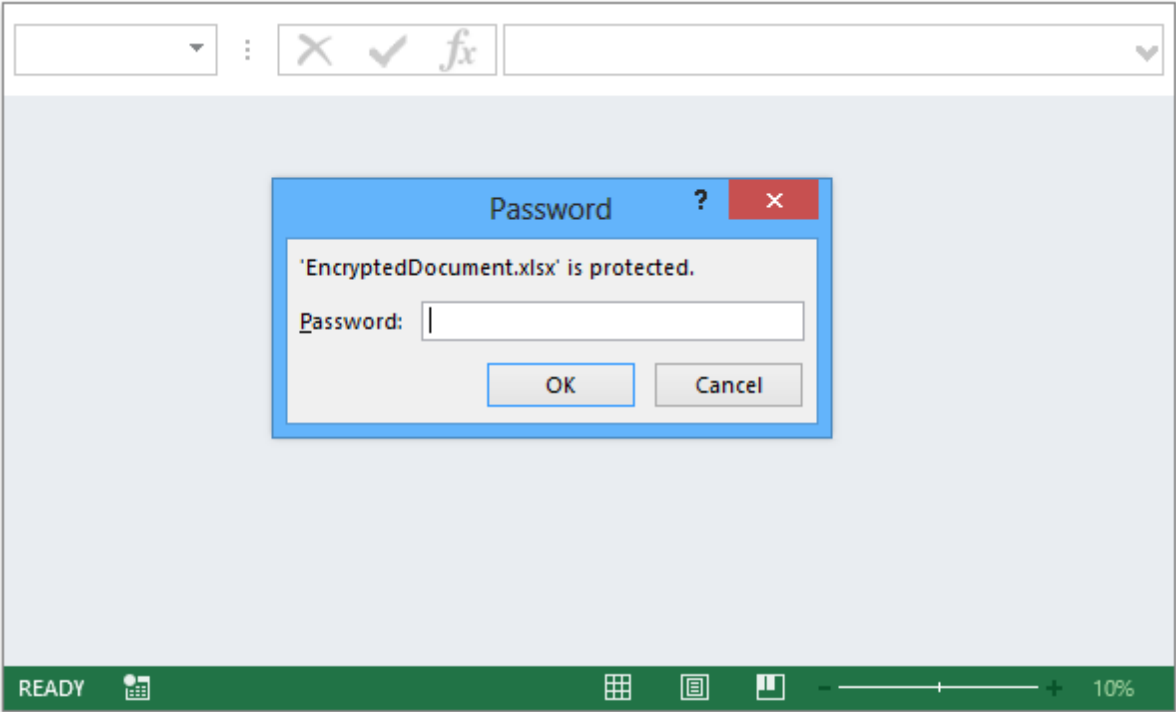
C#

```
(GeneralActions.cs)
// Create an exporter instance.
IXIExporter exporter = XlExport.CreateExporter(documentFormat);
// Specify encryption options.
// A workbook will be encrypted using the default encryption mechanism
// (agile encryption for XLSX files, and RC4 encryption for XLS files).
EncryptionOptions encryptionOptions = new EncryptionOptions();
// Specify the encryption password.
encryptionOptions.Password = "password";
// Create a new document and encrypt its contents.
using (IXIDocument document = exporter.CreateDocument(stream, encryptionOptions))
{
    // Specify the document culture.
    document.Options.Culture = CultureInfo.CurrentCulture;
}
```

Visual Basic

```
(GeneralActions.vb)
' Create an exporter instance.
Dim exporter As IXIExporter = XlExport.CreateExporter(documentFormat)
' Specify encryption options.
' A workbook will be encrypted using the default encryption mechanism
' (agile encryption for XLSX files, and RC4 encryption for XLS files).
Dim encryptionOptions As New EncryptionOptions()
' Specify the encryption password.
encryptionOptions.Password = "password"
' Create a new document and encrypt its contents.
Using document As IXIDocument = exporter.CreateDocument(stream, encryptionOptions)
    ' Specify the document culture.
    document.Options.Culture = CultureInfo.CurrentCulture
End Using
```

When end-users open the encrypted document in Microsoft® Excel®, the following dialog prompts them for a password:



How to: Specify Document Properties for a Workbook

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Workbooks](#) > [How to: Specify Document Properties for a Workbook](#)

The **Excel Export** Library allows you to set and edit the **document properties** (metadata) associated and stored with a workbook. You can either specify built-in properties or create your own custom properties.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

This example demonstrates how to specify the standard and custom document properties for a workbook using the `IXIDocument.Properties` and `XIDocumentProperties.Custom` properties, respectively.

Use the `IXIDocument.Properties` property to access the `XIDocumentProperties` object. This object provides a set of properties that allow you to specify basic information about a workbook, such as `XIDocumentProperties.Title`, `XIDocumentProperties.Author`, `XIDocumentProperties.Subject`, etc.

The `XIDocumentProperties.Custom` property allows you to get access to the `XIDocumentCustomProperties` object, which represents a storage of custom document properties. To set a value of a custom property with the specified name, use the `XIDocumentCustomProperties.Item` property. You can set a custom property to any object of the [String](#), [Double](#), [DateTime](#) or [Boolean](#) type. The specified object will be converted to the `XICustomPropertyValue` object and assigned to the custom property. To get the type of the data contained in a custom document property, use the `XICustomPropertyValue.Type` property.

C#

```
(MiscellaneousActions.cs)
// Create a new document.
using (IXIDocument document = exporter.CreateDocument(stream)) {
    document.Options.Culture = CultureInfo.CurrentCulture;
    // Set the built-in document properties.
    document.Properties.Title = "XL Export API: document properties example";
    document.Properties.Subject = "XL Export API";
    document.Properties.Keywords = "XL Export, document generation";
    document.Properties.Description = "How to set document properties using the XL Export API";
    document.Properties.Category = "Spreadsheet";
    document.Properties.Company = "DevExpress Inc.";
    // Set the custom document properties.
    document.Properties.Custom["Product Suite"] = "XL Export Library";
    document.Properties.Custom["Revision"] = 5;
    document.Properties.Custom["Date Completed"] = DateTime.Now;
    document.Properties.Custom["Published"] = true;
    // Generate data for the document.
    using (IXISheet sheet = document.CreateSheet()) {
        sheet.SkipRows(1);
        using (IXIRow row = sheet.CreateRow()) {
            row.SkipCells(1);
            using (IXICell cell = row.CreateCell()) {
                cell.Value = "You can view document properties using the File->Info->Properties->Advanced";
            }
        }
    }
}
```

Visual Basic

```
(MiscellaneousActions.vb)
' Create a new document.
Using document As IXlDocument = exporter.CreateDocument(stream)
    document.Options.Culture = CultureInfo.CurrentCulture
    ' Set the built-in document properties.
    document.Properties.Title = "XL Export API: document properties example"
    document.Properties.Subject = "XL Export API"
    document.Properties.Keywords = "XL Export, document generation"
    document.Properties.Description = "How to set document properties using the F"
    document.Properties.Category = "Spreadsheet"
    document.Properties.Company = "DevExpress Inc."
    ' Set the custom document properties.
    document.Properties.Custom("Product Suite") = "XL Export Library"
    document.Properties.Custom("Revision") = 5
    document.Properties.Custom("Date Completed") = Date.Now
    document.Properties.Custom("Published") = True
    ' Generate data for the document.
    Using sheet As IXlSheet = document.CreateSheet()
        sheet.SkipRows(1)
        Using row As IXlRow = sheet.CreateRow()
            row.SkipCells(1)
            Using cell As IXlCell = row.CreateCell()
                cell.Value = "You can view document properties using the F"
            End Using
        End Using
    End Using
End Using
```

To remove all custom document properties from a workbook, use the `XlDocumentCustomProperties.Clear` method.

Worksheets

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Worksheets](#)

This section contains the following examples:

- [How to: Create a New Worksheet](#)
- [How to: Set a Worksheet Name](#)
- [How to: Hide a Worksheet](#)
- [How to: Show and Hide Row and Column Headers](#)
- [How to: Show and Hide Gridlines](#)

How to: Create a New Worksheet

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Worksheets](#) > [How to: Create a New Worksheet](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

The example below demonstrates how to add a worksheet to a workbook. To do this, use the `IXIDocument.CreateSheet` method.

To specify a worksheet name, use the `IXISheet.Name` property. When naming a worksheet, take into account the constraints listed in the [How to: Set a Worksheet Name](#) document. If you do not specify a worksheet name, the default name `"SheetN"` is used, where *N* is a sequential number of a worksheet within a workbook.

C#

```
(GeneralActions.cs)
// Create a new document.
using (IXlDocument document = exporter.CreateDocument(stream)) {
    // Specify the document culture.
    document.Options.Culture = CultureInfo.CurrentCulture;
    // Create a new worksheet under the specified name.
    using (IXlSheet sheet = document.CreateSheet()) {
        sheet.Name = "Sales report";
    }
}
```

Visual Basic

```
(GeneralActions.vb)
' Create a new document.
Using document As IXlDocument = exporter.CreateDocument(stream)
    ' Specify the document culture.
    document.Options.Culture = CultureInfo.CurrentCulture
    ' Create a new worksheet under the specified name.
    Using sheet As IXlSheet = document.CreateSheet()
        sheet.Name = "Sales report"
    End Using
End Using
```

How to: Set a Worksheet Name

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Worksheets](#) > [How to: Set a Worksheet Name](#)

The example below demonstrates how to set a worksheet name. To do this, use the `IXISheet.Name` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
using (IXlSheet sheet = document.CreateSheet())
{
    // Specify the worksheet name.
    sheet.Name = "Sales report";
}
```

Visual Basic

```
Using sheet As IXlSheet = document.CreateSheet()
    ' Specify the worksheet name.
    sheet.Name = "Sales report"
End Using
```

When naming a worksheet, take into account the following constraints.

- The worksheet name must be unique.
- The worksheet name must not exceed 31 characters.
- The worksheet name must not contain the following symbols: \, /, ?, :, *, [or]
- The worksheet name must not start and end with a single quote (').
- The worksheet name must not be an empty string.

How to: Hide a Worksheet

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Worksheets](#) > [How to: Hide a Worksheet](#)

The example below demonstrates how to manage worksheet visibility in a workbook. To do this, use the `IXISheet.VisibleState` property.

Set this property to the `XISheetVisibleState.Hidden` value to hide a worksheet. End-users can show a hidden worksheet from the user interface (for example, when opening a workbook in Microsoft® Excel®). If you wish to prevent end-users from displaying hidden worksheets, mark a worksheet as "very hidden" by setting the `IXISheet.VisibleState` property to `XISheetVisibleState.VeryHidden`.

To restore the worksheet visibility, set the `IXISheet.VisibleState` property to the `XISheetVisibleState.Visible` enumeration member.

Important

A workbook must always contain at least one visible worksheet.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

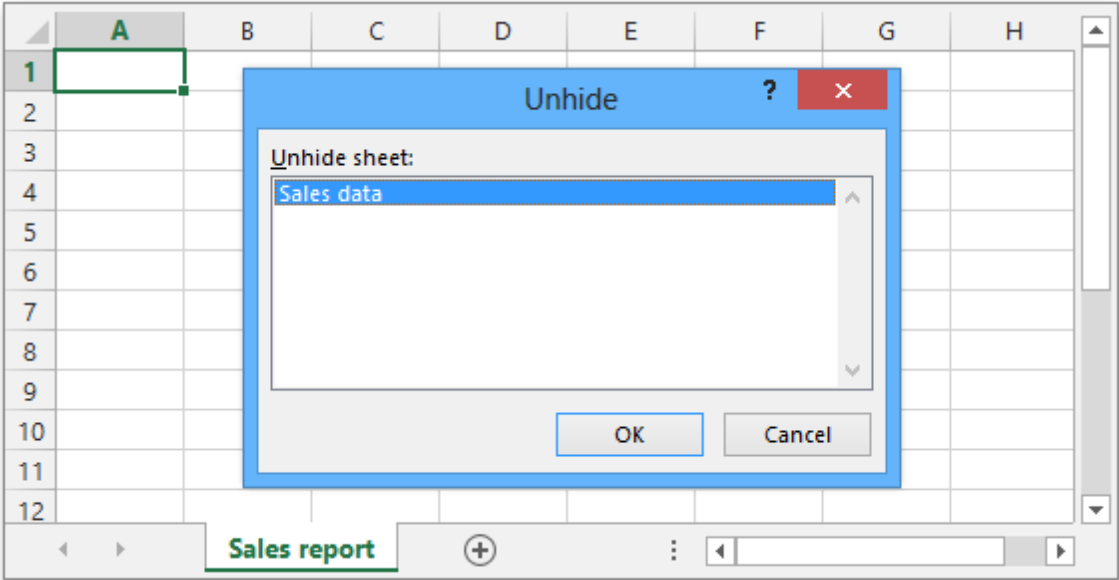
C#

```
(GeneralActions.cs)
// Create the first worksheet.
using (IXISheet sheet = document.CreateSheet()) {
    sheet.Name = "Sales report";
}
// Create the second worksheet and specify its visibility.
using (IXISheet sheet = document.CreateSheet()) {
    sheet.Name = "Sales data";
    sheet.VisibleState = XISheetVisibleState.Hidden;
}
```

Visual Basic

```
(GeneralActions.vb)
' Create the first worksheet.
Using sheet As IXISheet = document.CreateSheet()
    sheet.Name = "Sales report"
End Using
' Create the second worksheet and specify its visibility.
Using sheet As IXISheet = document.CreateSheet()
    sheet.Name = "Sales data"
    sheet.VisibleState = XISheetVisibleState.Hidden
End Using
```

The image below shows the result. The "Sales data" worksheet is hidden, but its visibility can be restored by an end-user from the user interface (the document is opened in Microsoft® Excel®).



How to: Show and Hide Row and Column Headers

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Worksheets](#) > [How to: Show and Hide Row and Column Headers](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

Each worksheet (IXISheet) consists of cells (IXICell) that are arranged in rows (IXIRow) and columns (IXIColumn). Each row and column has its own unique name (numbers "1", "2", "3", etc. are used for rows and letters A", "B", "C", etc. are used for columns). These unique names for rows and columns are displayed as headers at the left and at the top of a worksheet, respectively. To specify the visibility of row and column headers on a worksheet, use the IXISheetViewOptions.ShowRowColumnHeaders property of the IXISheetViewOptions object that controls worksheet display settings. It can be accessed using the IXISheet.ViewOptions property.

C#

```
(GeneralActions.cs)
// Create a worksheet.
using (IXISheet sheet = document.CreateSheet())
{
    // Hide row and column headers in the worksheet.
    sheet.ViewOptions.ShowRowColumnHeaders = false;
}
```

Visual Basic

```
(GeneralActions.vb)
' Create a worksheet.
Using sheet As IXISheet = document.CreateSheet()
    ' Hide row and column headers in the worksheet.
    sheet.ViewOptions.ShowRowColumnHeaders = False
End Using
```

The image below shows the appearance of the worksheet when row and column headers are shown, and when row and column headers are hidden (the workbook is opened in Microsoft® Excel®).

sheet.ViewOptions.ShowRowColumnHeaders = true

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					

Sheet1

READY

sheet.ViewOptions.ShowRowColumnHeaders = false

Sheet1

READY

How to: Show and Hide Gridlines

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Worksheets](#) > [How to: Show and Hide Gridlines](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

The example below demonstrates how to control the visibility of worksheet gridlines (the faint lines that separate rows and columns in a worksheet). To do this, use the `IXISheetViewOptions.ShowGridLines` property of the `IXISheetViewOptions` object that contains worksheet display settings and can be accessed using the `IXISheet.ViewOptions` property.

C#

```
(GeneralActions.cs)
// Create a worksheet.
using (IXISheet sheet = document.CreateSheet())
{
    // Hide gridlines on the worksheet.
    sheet.ViewOptions.ShowGridLines = false;
}
```

Visual Basic

```
(GeneralActions.vb)
' Create a worksheet.
Using sheet As IXISheet = document.CreateSheet()
    ' Hide gridlines on the worksheet.
    sheet.ViewOptions.ShowGridLines = False
End Using
```

The image below shows the worksheet when gridlines are displayed, and when gridlines are hidden (the workbook is opened in Microsoft® Excel®).

sheet.ViewOptions.ShowGridLines = true

sheet.ViewOptions.ShowGridLines = false

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					

Sheet1

+

READY

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					

Sheet1

+

READY

Rows and Columns

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Rows and Columns](#)

This section contains the following examples:

- [How to: Create a Column](#)
- [How to: Create a Row](#)
- [How to: Hide a Row or Column](#)
- [How to: Specify Row Height and Column Width](#)
- [How to: Freeze Rows and Columns](#)

How to: Create a Column

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Rows and Columns](#) > [How to: Create a Column](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

The example below demonstrates how to create and modify a column in a worksheet. To do this, use the `IXlSheet.CreateColumn` method. To specify where a created column should be located in a worksheet, pass a zero-based column index to the method as a parameter.

C#

```
// Create a new worksheet.
using (IXlSheet sheet = document.CreateSheet())
{
    // Create the column "A" and set its width to 100 pixels.
    using (IXlColumn column = sheet.CreateColumn())
    {
        column.WidthInPixels = 100;
    }
    // Create the column D and set its width to 24.5 characters.
    using (IXlColumn column = sheet.CreateColumn(3))
    {
        column.WidthInCharacters = 24.5f;
    }
}
```

Visual Basic

```
' Create a new worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Create the column "A" and set its width to 100 pixels.
    Using column As IXlColumn = sheet.CreateColumn()
        column.WidthInPixels = 100
    End Using
    ' Create the column D and set its width to 24.5 characters.
    Using column As IXlColumn = sheet.CreateColumn(3)
        column.WidthInCharacters = 24.5F
    End Using
End Using
```

Note

The number of columns in a worksheet is permanently fixed and depends on the output file format (16,384 columns for **XLSX** and **CSV** files, and 256 columns for **XLS** files). Note that you can use the `IXlDocumentOptions.MaxColumnCount` property to obtain the maximum number of columns supported by the file format to which the document is exported.

Important

Columns in a worksheet should be created prior to rows and cells. If you try to create a column after or during the row generation, an **InvalidOperationException** exception will be thrown.

How to: Create a Row

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Rows and Columns](#) > [How to: Create a Row](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

The example below demonstrates how to create and modify a row in a worksheet. To do this, use the `IXISheet.CreateRow` method. To specify where a created row should be located in a worksheet, pass a zero-based row index to the method as a parameter.

C#

```
// Create a new worksheet.
using (IXISheet sheet = document.CreateSheet())
{
    // Create the third row and set its height to 40 pixels.
    using (IXIRow row = sheet.CreateRow(2))
    {
        row.HeightInPixels = 40;
    }
}
```

Visual Basic

```
' Create a new worksheet.
Using sheet As IXISheet = document.CreateSheet()
    ' Create the third row and set its height to 40 pixels.
    Using row As IXIRow = sheet.CreateRow(2)
        row.HeightInPixels = 40
    End Using
End Using
```

Note

The number of rows in a worksheet is permanently fixed and depends on the output file format (1,048,576 rows for **XLSX** and **CSV** files, and 65,536 rows for **XLS** files). Note that you can use the `IXIDocumentOptions.MaxRowCount` property to obtain the maximum number of rows supported by the file format to which the document is exported.

How to: Hide a Row or Column

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Rows and Columns](#) > [How to: Hide a Row or Column](#)

The example below demonstrates how to control the visibility of rows and columns in a worksheet using the `IXlRow.IsHidden` and `IXlColumn.IsHidden` properties.

Note

You can also hide a row or column in a worksheet by setting the row height (`IXlRow.HeightInPixels`) or column width (`IXlColumn.WidthInCharacters` or `IXlColumn.WidthInPixels`) to 0, respectively (see the [How to: Specify Row Height and Column Width](#) document).

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=T253492>.

C#

```
// Create a new worksheet.
using (IXlSheet sheet = document.CreateSheet())
{
    // Hide the column B in the worksheet.
    using (IXlColumn column = sheet.CreateColumn(1))
    {
        column.IsHidden = true;
    }
    // Hide the third row in the worksheet.
    using (IXlRow row = sheet.CreateRow(2))
    {
        row.IsHidden = true;
    }
}
```

Visual Basic

```
' Create a new worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Hide the column B in the worksheet.
    Using column As IXlColumn = sheet.CreateColumn(1)
        column.IsHidden = True
    End Using
    ' Hide the third row in the worksheet.
    Using row As IXlRow = sheet.CreateRow(2)
        row.IsHidden = True
    End Using
End Using
```

See Also

[How to: Specify Row Height and Column Width](#)

How to: Specify Row Height and Column Width

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Rows and Columns](#) > [How to: Specify Row Height and Column Width](#)

Columns

To specify a column width in characters of the default font defined by the *Normal* style, use the column's `IXIColumn.WidthInCharacters` property.

To specify a column width in pixels, use the column's `IXIColumn.WidthInPixels` property.

Note

If a column width is set to 0, the column is hidden. You can also use the `IXIColumn.IsHidden` property to hide a column or display the hidden column again (see the [How to: Hide a Row or Column](#) document).

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

// Create a new worksheet.
using (IXISheet sheet = document.CreateSheet())
{
 // Create the column A and set its width to 100 pixels.
 using (IXIColumn column = sheet.CreateColumn())
 {
 column.WidthInPixels = 100;
 }
 // Create the column B and set its width to 24.5 characters.
 using (IXIColumn column = sheet.CreateColumn())
 {
 column.WidthInCharacters = 24.5f;
 }
}

Visual Basic

' Create a new worksheet.
Using sheet As IXISheet = document.CreateSheet()
 ' Create the column A and set its width to 100 pixels.
 Using column As IXIColumn = sheet.CreateColumn()
 column.WidthInPixels = 100
 End Using
 ' Create the column B and set its width to 24.5 characters.
 Using column As IXIColumn = sheet.CreateColumn()
 column.WidthInCharacters = 24.5f
 End Using
End Using

Rows

To specify a row height in pixels, use the row's `IXIRow.HeightInPixels` property.

To specify a row height in points, use the row's `IXIRow.HeightInPoints` property.

Note

If the row height is set to 0, the row is hidden. You can also use the `IXIRow.IsHidden` property to hide a row or display the hidden row again (see the [How to: Hide a Row or Column](#) document).

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
// Create a new worksheet.
using (IXlSheet sheet = document.CreateSheet())
{
    // Create the third row and set its height to 40 pixels.
    using (IXlRow row = sheet.CreateRow(2))
    {
        row.HeightInPixels = 40;
    }
}
```

Visual Basic

```
' Create a new worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Create the third row and set its height to 40 pixels.
    Using row As IXlRow = sheet.CreateRow(2)
        row.HeightInPixels = 40
    End Using
End Using
```

See Also

[How to: Hide a Row or Column](#)

How to: Freeze Rows and Columns

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Rows and Columns](#) > [How to: Freeze Rows and Columns](#)

You can lock a number of top rows and left columns to keep an area of the worksheet permanently visible while scrolling another worksheet area. To do this, specify the position of a cell below the frozen rows and to the right of the frozen columns by creating the `XlCellPosition` object and assign it to the `IXISheet.SplitPosition` property.

To freeze the first row, use the following code:

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(PageViewAndLayoutActions.cs)
// Freeze the first row in the worksheet.
sheet.SplitPosition = new XlCellPosition(0, 1);
```

Visual Basic

```
(PageViewAndLayoutActions.vb)
' Freeze the first row in the worksheet.
sheet.SplitPosition = New XlCellPosition(0, 1)
```

To freeze the first column, use the following code:

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(PageViewAndLayoutActions.cs)
// Freeze the first column in the worksheet.
sheet.SplitPosition = new XlCellPosition(1, 0);
```

Visual Basic

```
(PageViewAndLayoutActions.vb)
' Freeze the first column in the worksheet.
sheet.SplitPosition = New XlCellPosition(1, 0)
```

To freeze the pane which contains one topmost row and one leftmost column, use the following code:

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(PageViewAndLayoutActions.cs)
// Freeze the first column and the first row.
sheet.SplitPosition = new XlCellPosition(1, 1);
```

Visual Basic

```
(PageViewAndLayoutActions.vb)  
' Freeze the first column and the first row.  
sheet.SplitPosition = New XlCellPosition(1, 1)
```

Cells

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Cells](#)

This section contains the following examples:

- [How to: Create a Worksheet Cell and Set Its Value](#)
- [How to: Merge Cells or Split Merged Cells](#)
- [How to: Add a Hyperlink to a Cell](#)

How to: Create a Worksheet Cell and Set Its Value

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Cells](#) > [How to: Create a Worksheet Cell and Set Its Value](#)

The example below demonstrates how to add cells to a worksheet. To do this, call the `IXIRow.CreateCell` method of the row where you wish to create a cell. If required, pass the following parameter: the zero-based index of the column where the new cell should be located. As a result, the cell will be created at the intersection of the specified row and column.

Each cell in a worksheet can contain a single piece of data - the cell value specified by the `XIVariantValue` object. To set a cell value, assign the required value to the `IXICell.Value` property. A cell value can be of one of the following types: **numeric**, **text**, **Boolean** or **error**. Data values of these types can have various display formats. For example, a numeric value can be displayed as a decimal number, a percentage or currency value, a date or time value, etc. To specify the cell number format, use the `IXICell.Formatting` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

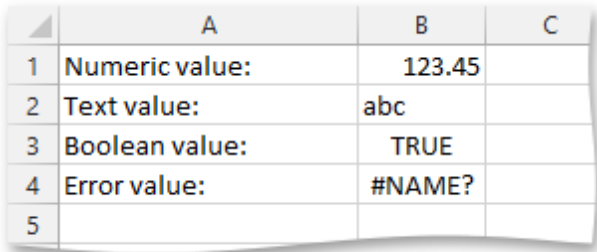
```
(GeneralActions.cs)
// Create a worksheet.
using (IXlSheet sheet = document.CreateSheet()) {
    // Create the column A and set its width.
    using (IXlColumn column = sheet.CreateColumn()) {
        column.WidthInPixels = 150;
    }
    // Create the first row.
    using (IXlRow row = sheet.CreateRow()) {
        // Create the cell A1 and set its value.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Numeric value:";
        }
        // Create the cell B1 and assign the numeric value to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = 123.45;
        }
    }
    // Create the second row.
    using (IXlRow row = sheet.CreateRow()) {
        // Create the cell A2 and set its value.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Text value:";
        }
        // Create the cell B2 and assign the text value to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "abc";
        }
    }
    // Create the third row.
    using (IXlRow row = sheet.CreateRow()) {
        // Create the cell A3 and set its value.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Boolean value:";
        }
        // Create the cell B3 and assign the boolean value to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = true;
        }
    }
    // Create the fourth row.
    using (IXlRow row = sheet.CreateRow()) {
        // Create the cell A4 and set its value.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Error value:";
        }
        // Create the cell B4 and assign an error value to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = XlVariantValue.ErrorName;
        }
    }
}
```

Visual Basic

```
(GeneralActions.vb)
' Create a worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Create the column A and set its width.
    Using column As IXlColumn = sheet.CreateColumn()
        column.WidthInPixels = 150
    End Using
    ' Create the first row.
    Using row As IXlRow = sheet.CreateRow()
        ' Create the cell A1 and set its value.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Numeric value:"
        End Using
        ' Create the cell B1 and assign the numeric value to it.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = 123.45
        End Using
    End Using
    ' Create the second row.
    Using row As IXlRow = sheet.CreateRow()
        ' Create the cell A2 and set its value.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Text value:"
        End Using
        ' Create the cell B2 and assign the text value to it.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "abc"
        End Using
    End Using
    ' Create the third row.
    Using row As IXlRow = sheet.CreateRow()
        ' Create the cell A3 and set its value.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Boolean value:"
        End Using
        ' Create the cell B3 and assign the boolean value to it.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = True
        End Using
    End Using
    ' Create the fourth row.
    Using row As IXlRow = sheet.CreateRow()
        ' Create the cell A4 and set its value.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Error value:"
        End Using
        ' Create the cell B4 and assign an error value to it.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = XlVariantValue.ErrorName
        End Using
    End Using
End Using
```

End Using

The image below shows the result (the workbook is opened in Microsoft® Excel®).

A screenshot of an Excel spreadsheet with three columns labeled A, B, and C. The rows are numbered 1 through 5. Row 1 contains 'Numeric value:' in A and '123.45' in B. Row 2 contains 'Text value:' in A and 'abc' in B. Row 3 contains 'Boolean value:' in A and 'TRUE' in B. Row 4 contains 'Error value:' in A and '#NAME?' in B. Row 5 is empty.

	A	B	C
1	Numeric value:	123.45	
2	Text value:	abc	
3	Boolean value:	TRUE	
4	Error value:	#NAME?	
5			

How to: Merge Cells or Split Merged Cells

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Cells](#) > [How to: Merge Cells or Split Merged Cells](#)

This example illustrates how to use the **Excel Export** library to merge and unmerge cells in a worksheet.

Merge Cells

To merge adjacent cells in a worksheet into a single cell, pass the range of cells to be merged to the IXIMergedCells.Add method of the IXIMergedCells object, which represents the collection of all merged cells in a worksheet and can be accessed from the IXISheet.MergedCells property.

Note

When you merge a cell range, the content and format settings of only the top-left cell will appear in the resulting merged cell. The data contained in other cells of the original range will be lost.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

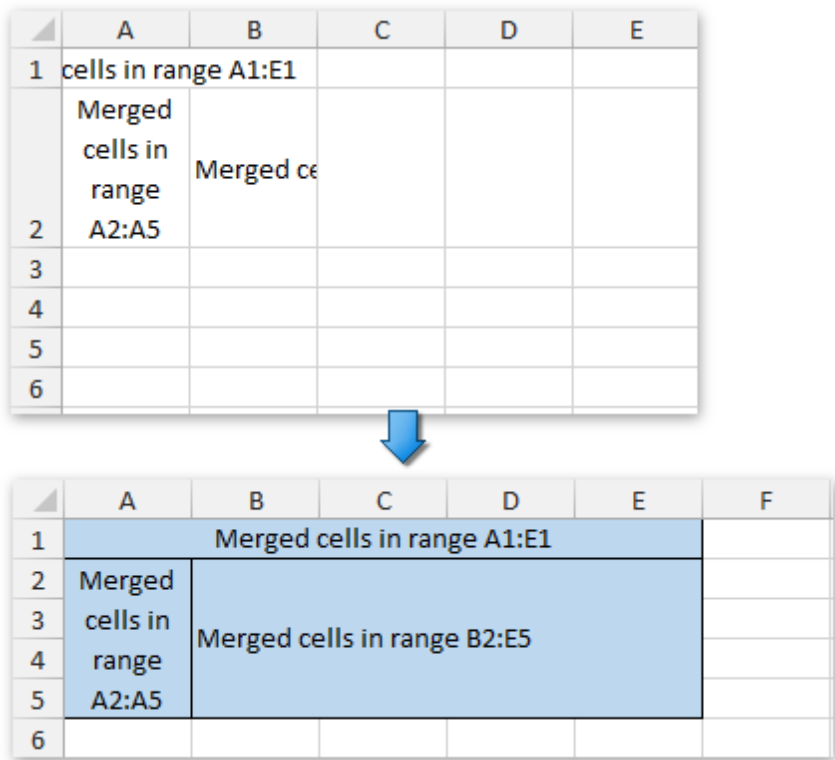
C#

(GeneralActions.cs)
// Merge cells contained in the range A1:E1.
sheet.MergedCells.Add(XlCellRange.FromLTRB(0, 0, 4, 0));
// Merge cells contained in the range A2:A5.
sheet.MergedCells.Add(XlCellRange.FromLTRB(0, 1, 0, 4));
// Merge cells contained in the range B2:E5.
sheet.MergedCells.Add(XlCellRange.FromLTRB(1, 1, 4, 4));

Visual Basic

(GeneralActions.vb)
' Merge cells contained in the range A1:E1.
sheet.MergedCells.Add(XlCellRange.FromLTRB(0, 0, 4, 0))
' Merge cells contained in the range A2:A5.
sheet.MergedCells.Add(XlCellRange.FromLTRB(0, 1, 0, 4))
' Merge cells contained in the range B2:E5.
sheet.MergedCells.Add(XlCellRange.FromLTRB(1, 1, 4, 4))

The image below illustrates the result of code execution (the workbook is opened in Microsoft® Excel®).



Important

By default, you cannot merge cells in a range that overlap existing merged cells in a worksheet. When you call the `IXIMergedCells.Add` method, it checks whether the cell range you wish to merge intersects other merged cells and throws a [ArgumentException](#) if it does. However, if there are many merged cells in your document, this checking operation may slow down the process of merging cells. To disable the internal checking of intersecting merged cells, and thereby speed up the document generation, call the `IXIMergedCells.Add` method overload with the `checkOverlap` parameter set to **false**. Keep in mind that in this case, you may get an invalid document if there are any merged cells that intersect other merged cells in a worksheet.

Split Merged Cells

You can split a merged cell into separate cells again by deleting the merged cells from the `IXIMergedCells` collection. To do this, use the `IXIMergedCells.Remove` or `IXIMergedCells.RemoveAt` method as shown in the example below.

C#

```
// Split the merged cell in the range A1:E1.
sheet.MergedCells.RemoveAt(0);
// Split the merged cell in the range A2:A5.
sheet.MergedCells.Remove(XlCellRange.FromLTRB(0, 1, 0, 4));
```

Visual Basic

```
' Split the merged cell in the range A1:E1.
sheet.MergedCells.RemoveAt(0)
' Split the merged cell in the range A2:A5.
sheet.MergedCells.Remove(XlCellRange.FromLTRB(0, 1, 0, 4))
```

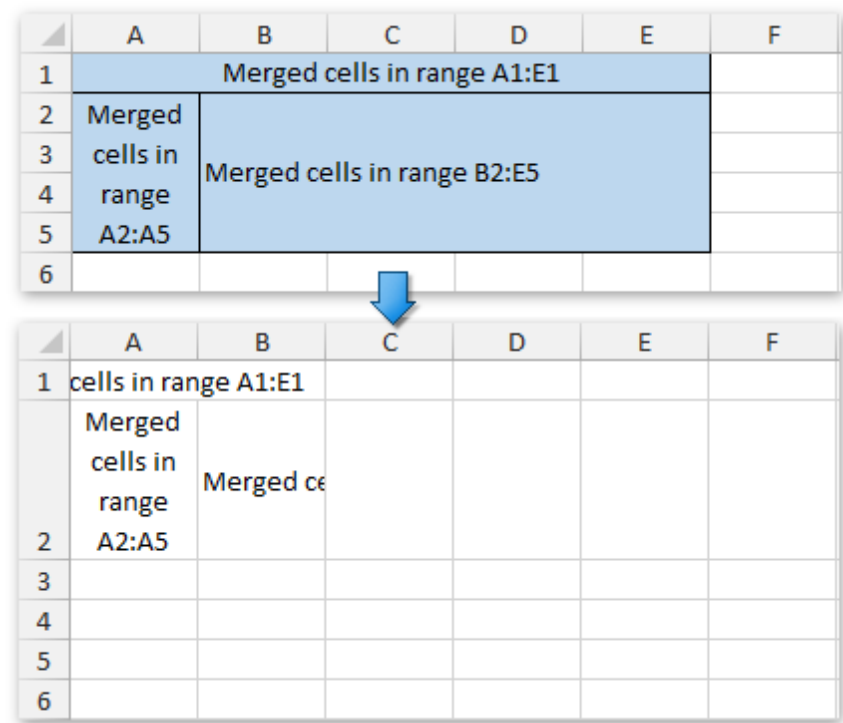
To split all merged cells in the worksheet at once, call the `IXIMergedCells.Clear` method.

C#

```
// Split all merged cells in the worksheet.
sheet.MergedCells.Clear();
```

```
Visual Basic
' Split all merged cells in the worksheet.
sheet.MergedCells.Clear()
```

The image below shows how cells are split in a worksheet (the workbook is opened in Microsoft® Excel®). Note that the content of the merged cell appears in the top-left cell of the range of split cells.



How to: Add a Hyperlink to a Cell

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Cells](#) > [How to: Add a Hyperlink to a Cell](#)

The **Excel Export Library** allows you to create hyperlinks represented by the `XIHyperlink` objects. All hyperlinks in a worksheet are contained in a collection returned by the `IXISheet.Hyperlinks` property.

To create a hyperlink, follow the steps below:

1. Initialize a new instance of the `XIHyperlink` class using the default `XIHyperlink.XIHyperlink` constructor.
2. Specify the cell or cell range to which the hyperlink should be attached by using the `XIHyperlink.Reference` property.
3. Use the `XIHyperlinkBase.TargetUri` property to specify the hyperlink destination. The following table lists possible locations to which hyperlinks can refer.

Hyperlink Destination	TargetUri Value	Example
Place in the current workbook	A cell reference or defined name preceded by the worksheet name.	"#Sheet1!B4"
Place in an external workbook	A cell reference or defined name preceded by the path to a file, the workbook and worksheet names. If a path to the destination workbook is not specified, the path relative to the location of the current workbook will be used.	"D:\\Shared\\Expenses.xlsx#Sheet1!C5" "Document.xls#Sheet1!B4"
Web page	A web address.	"http://www.devexpress.com/"
E-mail address	An e-mail address preceded by the "mailto:" prefix.	mailto:support@devexpress.com
Existing File or Directory	A path to a file and file name, or a path to a directory. A path can be absolute or relative to the directory where the current workbook is located.	"D:\\Pictures\\Image1.png" "..\\..\\..\\MyBook.xlsx"

4. Add a newly created hyperlink to the collection of hyperlinks contained in a worksheet.

If a cell range to which you added a hyperlink does not contain any value, you can provide a **display text** for it to indicate that this cell range has a hyperlink attached. To do this, [create a cell](#) that should contain hyperlink text. Normally, it is the top-left cell of the range associated with a hyperlink. Set the cell's `IXICell.Value` property to the string value that specifies the text you wish to use to represent the hyperlink. Format the specified text as a hyperlink by setting the `IXICell.Formatting` property to the `XICellFormatting.Hyperlink` object.

The code snippet below illustrates how to create a hyperlink to cells located in the same and external workbooks, and a web page.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(MiscellaneousActions.cs)
// Create a worksheet.
using (IXlSheet sheet = document.CreateSheet()) {
    using (IXlColumn column = sheet.CreateColumn()) {
        column.WidthInPixels = 300;
    }
    // Create a hyperlink to a cell in the current workbook.
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Local link";
            cell.Formatting = XlCellFormatting.Hyperlink;
            XlHyperlink hyperlink = new XlHyperlink();
            hyperlink.Reference = new XlCellRange(new XlCellPosition(cell.ColumnIndex, cell.RowIndex));
            hyperlink.TargetUri = "#Sheet1!C5";
            sheet.Hyperlinks.Add(hyperlink);
        }
    }
    // Create a hyperlink to a cell located in the external workbook.
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "External file link";
            cell.Formatting = XlCellFormatting.Hyperlink;
            XlHyperlink hyperlink = new XlHyperlink();
            hyperlink.Reference = new XlCellRange(new XlCellPosition(cell.ColumnIndex, cell.RowIndex));
            hyperlink.TargetUri = "linked.xlsx#Sheet1!C5";
            sheet.Hyperlinks.Add(hyperlink);
        }
    }
    // Create a hyperlink to a web page.
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "External URI";
            cell.Formatting = XlCellFormatting.Hyperlink;
            XlHyperlink hyperlink = new XlHyperlink();
            hyperlink.Reference = new XlCellRange(new XlCellPosition(cell.ColumnIndex, cell.RowIndex));
            hyperlink.TargetUri = "http://www.devexpress.com";
            sheet.Hyperlinks.Add(hyperlink);
        }
    }
}
```

Visual Basic

```
(MiscellaneousActions.vb)
' Create a worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    Using column As IXlColumn = sheet.CreateColumn()
        column.WidthInPixels = 300
    End Using
    ' Create a hyperlink to a cell in the current workbook.
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Local link"
            cell.Formatting = XlCellFormatting.Hyperlink
            Dim hyperlink As New XlHyperlink()
            hyperlink.Reference = New XlCellRange(New XlCellPosition(cell.
            hyperlink.TargetUri = "#Sheet1!C5"
            sheet.Hyperlinks.Add(hyperlink)
        End Using
    End Using
    ' Create a hyperlink to a cell located in the external workbook.
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "External file link"
            cell.Formatting = XlCellFormatting.Hyperlink
            Dim hyperlink As New XlHyperlink()
            hyperlink.Reference = New XlCellRange(New XlCellPosition(cell.
            hyperlink.TargetUri = "linked.xlsx#Sheet1!C5"
            sheet.Hyperlinks.Add(hyperlink)
        End Using
    End Using
    ' Create a hyperlink to a web page.
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "External URI"
            cell.Formatting = XlCellFormatting.Hyperlink
            Dim hyperlink As New XlHyperlink()
            hyperlink.Reference = New XlCellRange(New XlCellPosition(cell.
            hyperlink.TargetUri = "http://www.devexpress.com"
            sheet.Hyperlinks.Add(hyperlink)
        End Using
    End Using
End Using
```

Formulas

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formulas](#)

To create a [formula](#) in a cell, use the `IXICell.SetFormula` method. This method enables you to specify a formula in different ways:

- Use textual representation.

The most common and straightforward way. However, it is not the fastest method to set a formula, because it requires a formula parser (a **DevExpress.Export.Xl.XlFormulaParser** descendant, which should be supplied in the `XIExport.CreateExporter` method). The XL Export library writes data directly to the output stream in consecutive order, and you cannot evaluate formulas containing named ranges or custom functions. That is the reason why named range and custom functions cannot be written in .XLS (Excel 97-2003) format and in the .XLSX (OpenXml) format when the formula parser is specified.

Tip

You can use any string (even composed incorrectly or containing undefined names) as a formula when exporting to the .XLSX (OpenXml) format without a parser because formulas are not parsed and validated.

- Use `IXIFormulaParameter` object.

This is the recommended way to create common formulas containing built-in functions.

- Compose a formula from expression tokens.

Tokens are the descendants of the **DevExpress.Export.Xl.XlPtgBase** class ("parsed things"), arranged in reverse Polish notation. The method originates from .XLS file structure and is too complex for most cases.

This section contains the following examples which illustrate the use of the methods described above.

- [How to: Create a Cell Formula](#)
- [How to: Create Shared Formulas](#)
- [How to: Create Subtotals](#)

How to: Create a Cell Formula

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formulas](#) > [How to: Create a Cell Formula](#)

This example demonstrates how to create cell formulas using three different methods.

Formula from Textual Representation

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

In this code, the exporter instance is created with the `XIExport.CreateExporter` method which also creates the formula parser. A worksheet contains a heading row and four rows populated with data. The last column in each data row contains a formula created by supplying a formula string to the `IXICell.SetFormula` method.

C#

```

(FormulaActions.cs)
// Create an exporter instance.
IXlExporter exporter = XlExport.CreateExporter(documentFormat, new XlFormulaParser());
// Create a new document.
using (IXlDocument document = exporter.CreateDocument(stream)) {
    document.Options.Culture = CultureInfo.CurrentCulture;
    // Create a worksheet.
    using (IXlSheet sheet = document.CreateSheet()) {
        // Create worksheet columns and set their widths.
        for (int i = 0; i < 4; i++) {
            using (IXlColumn column = sheet.CreateColumn()) {
                column.WidthInPixels = 80;
            }
        }
        // Generate data for the document.
        string[] header = new string[] { "Description", "QTY", "Price", "Amount" };
        string[] product = new string[] { "Camembert", "Gorgonzola", "Mascarpone", "Mozzarella" };
        int[] qty = new int[] { 12, 15, 25, 10 };
        double[] price = new double[] { 23.25, 15.50, 12.99, 8.95 };
        double discount = 0.2;
        // Create the header row.
        using (IXlRow row = sheet.CreateRow()) {
            for (int i = 0; i < 4; i++) {
                using (IXlCell cell = row.CreateCell()) {
                    cell.Value = header[i];
                }
            }
        }
        // Create data rows using string formulas.
        for (int i = 0; i < 4; i++) {
            using (IXlRow row = sheet.CreateRow()) {
                using (IXlCell cell = row.CreateCell()) {
                    cell.Value = product[i];
                }
                using (IXlCell cell = row.CreateCell()) {
                    cell.Value = qty[i];
                }
                using (IXlCell cell = row.CreateCell()) {
                    cell.Value = price[i];
                }
                using (IXlCell cell = row.CreateCell()) {
                    // Set the formula to calculate the amount
                    // applying 20% quantity discount on orders more than 15 items.
                    cell.SetFormula(String.Format("=IF(B{0}>15,C{0}*B{0}*(1-{1}),C{0}*B{0})", i + 2, disc
                }
            }
        }
    }
}

```

Visual Basic

```

(FormulaActions.vb)
' Create an exporter instance.
Dim exporter As IXlExporter = XlExport.CreateExporter(documentFormat, New XlFormulaParser())
' Create a new document.
Using document As IXlDocument = exporter.CreateDocument(stream)
    document.Options.Culture = CultureInfo.CurrentCulture
    ' Create a worksheet.
    Using sheet As IXlSheet = document.CreateSheet()
        ' Create worksheet columns and set their widths.
        For i As Integer = 0 To 3
            Using column As IXlColumn = sheet.CreateColumn()
                column.WidthInPixels = 80
            End Using
        Next i
        ' Generate data for the document.
        Dim header() As String = { "Description", "QTY", "Price", "Amount" }
        Dim product() As String = { "Camembert", "Gorgonzola", "Mascarpone", "Mozzarella" }
        Dim qty() As Integer = { 12, 15, 25, 10 }
        Dim price() As Double = { 23.25, 15.50, 12.99, 8.95 }
        Dim discount As Double = 0.2
        ' Create the header row.
        Using row As IXlRow = sheet.CreateRow()
            For i As Integer = 0 To 3
                Using cell As IXlCell = row.CreateCell()
                    cell.Value = header(i)
                End Using
            Next i
        End Using
        ' Create data rows using string formulas.
        For i As Integer = 0 To 3
            Using row As IXlRow = sheet.CreateRow()
                Using cell As IXlCell = row.CreateCell()
                    cell.Value = product(i)
                End Using
                Using cell As IXlCell = row.CreateCell()
                    cell.Value = qty(i)
                End Using
                Using cell As IXlCell = row.CreateCell()
                    cell.Value = price(i)
                End Using
                Using cell As IXlCell = row.CreateCell()
                    ' Set the formula to calculate the amount
                    ' applying 20% quantity discount on orders more than 15 items.
                    cell.SetFormula(String.Format("=IF(B{0}>15,C{0}*B{0}*(1-{1}),C{0}*B{0})", i + 2, disc
                End Using
            End Using
        Next i
    End Using
End Using

```

Formula from Expression Elements

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

This code snippet creates an IXlFormulaParameter expression from a combination of constants, operators and functions. Constants are transformed into the IXlFormulaParameter objects with the XlFunc.Param method. Operators are static methods of the XlOper object and functions are static methods of the XlFunc object.

When an expression is created, the `IXlCell.SetFormula` method is used to enter expression into a worksheet cell as the cell formula.

C#

```
(FormulaActions.cs)
// Create the total row using IXlFormulaParameter.
using (IXlRow row = sheet.CreateRow()) {
    row.SkipCells(2);
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Total:";
        cell.ApplyFormatting(totalRowFormatting);
    }
    using (IXlCell cell = row.CreateCell()) {
        // Set the formula to calculate the total amount plus 10 handling
        // =SUM($D$2:$D$5)+10
        IXlFormulaParameter const10 = XlFunc.Param(10);
        IXlFormulaParameter sumAmountFunction = XlFunc.Sum(XlCellRange.From
        cell.SetFormula(XlOper.Add(sumAmountFunction, const10));
        cell.ApplyFormatting(totalRowFormatting);
    }
}
```

Visual Basic

```
(FormulaActions.vb)
' Create the total row using IXlFormulaParameter.
Using row As IXlRow = sheet.CreateRow()
    row.SkipCells(2)
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Total:"
        cell.ApplyFormatting(totalRowFormatting)
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Set the formula to calculate the total amount plus 10 handling f
        ' =SUM($D$2:$D$5)+10
        Dim const10 As IXlFormulaParameter = XlFunc.Param(10)
        Dim sumAmountFunction As IXlFormulaParameter = XlFunc.Sum(XlCellRa
        cell.SetFormula(XlOper.Add(sumAmountFunction, const10))
        cell.ApplyFormatting(totalRowFormatting)
    End Using
End Using
```

Formula from Tokens

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

This code snippet show how to create an expression "from scratch" by adding formula tokens (aka PTGs, "parsed things") to the `XlExpression` instance which is a `System.Collections.Generic.List`1[[DevExpress.Export.Xl.XlPtgBase]]` list of tokens

arranged in Reverse-Polish Notation order. Subsequently, the expression is passed to the `IXlCell.SetFormula` method to specify a cell formula.

C#

```
(FormulaActions.cs)
// Create a formula using XlExpression.
using (IXlRow row = sheet.CreateRow()) {
    row.SkipCells(2);
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Mean value:";
        cell.ApplyFormatting(totalRowFormatting);
    }
    using (IXlCell cell = row.CreateCell()) {
        // Set the formula to calculate the mean value.
        // =$D$6/4
        XlExpression expression = new XlExpression();
        expression.Add(new XlPtgRef(new XlCellPosition(cell.ColumnIndex, r
        expression.Add(new XlPtgInt(row.RowIndex - 2));
        expression.Add(new XlPtgBinaryOperator(XlPtgTypeCode.Div));
        cell.SetFormula(expression);
        cell.ApplyFormatting(totalRowFormatting);
    }
}
```

Visual Basic

```
(FormulaActions.vb)
' Create a formula using XlExpression.
Using row As IXlRow = sheet.CreateRow()
    row.SkipCells(2)
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Mean value:"
        cell.ApplyFormatting(totalRowFormatting)
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Set the formula to calculate the mean value.
        ' =$D$6/4
        Dim expression As New XlExpression()
        expression.Add(New XlPtgRef(New XlCellPosition(cell.ColumnIndex, r
        expression.Add(New XlPtgInt(row.RowIndex - 2))
        expression.Add(New XlPtgBinaryOperator(XlPtgTypeCode.Div))
        cell.SetFormula(expression)
        cell.ApplyFormatting(totalRowFormatting)
    End Using
End Using
```

How to: Create Shared Formulas

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formulas](#) > [How to: Create Shared Formulas](#)

A shared formula consists of a formula in one cell marked as shared and the cells referenced by the shared formula. The referenced cells obtain their own version of the formula, so if cell D2 has the formula B2*C2, then cell D3 would have the formula B3*C3 etc.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

It takes two steps to build a shared formula. First, add a formula to the first cell in a range - it is the host formula. Next, add a reference to the host formula to any other cell of the range. The `IXlCell.SetSharedFormula` method with different parameters serves both steps, as illustrated in the code below.

C#

(FormulaActions.cs)
// Create data rows.
for (int i = 0; i < 4; i++) {
 using (IXlRow row = sheet.CreateRow()) {
 using (IXlCell cell = row.CreateCell()) {
 cell.Value = product[i];
 }
 using (IXlCell cell = row.CreateCell()) {
 cell.Value = qty[i];
 }
 using (IXlCell cell = row.CreateCell()) {
 cell.Value = price[i];
 }
 using (IXlCell cell = row.CreateCell()) {
 // Use the shared formula to calculate the amount per product.
 if (i == 0)
 cell.SetSharedFormula("B2*C2", XlCellRange.FromLTRB(3, 1, 3, 4));
 else
 cell.SetSharedFormula(new XlCellPosition(3, 1));
 }
 }
}
}

Visual Basic

```
(FormulaActions.vb)
' Create data rows.
For i As Integer = 0 To 3
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = product(i)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = qty(i)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = price(i)
        End Using
        Using cell As IXlCell = row.CreateCell()
            ' Use the shared formula to calculate the amount per product.
            If i = 0 Then
                cell.SetSharedFormula("B2*C2", XlCellRange.FromLTRB(3, 1,
            Else
                cell.SetSharedFormula(New XlCellPosition(3, 1))
            End If
        End Using
    End Using
Next i
```



```
(FormulaActions.vb)
' Create the grand total row.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Grand Total"
        cell.ApplyFormatting(totalRowFormatting)
        cell.Formatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeC
    End Using
    For j As Integer = 0 To 4
        Using cell As IXlCell = row.CreateCell()
            ' Use the SUBTOTAL function to calculate grand total sales for
            cell.SetFormula(XlFunc.Subtotal(XlCellRange.FromLTRB(j + 1, st
            cell.ApplyFormatting(totalRowFormatting)
        End Using
    Next j
End Using
```

Formatting

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#)

This section contains the following examples:

- [How to: Format a Cell](#)
- [How to: Apply Predefined Formatting to a Cell](#)
- [How to: Apply Themed Formatting to a Cell](#)
- [How to: Change Cell Background Color](#)
- [How to: Configure Cell Font Settings](#)
- [How to: Add Cell Borders](#)
- [How to: Align Cell Content](#)
- [How to: Specify Number Format for Cell Content](#)
- [How to: Apply Rich Formatting to the Cell Text](#)

How to: Format a Cell

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Format a Cell](#)

The proper formatting of worksheet cells improves document appearance and makes document data easier to read and understand. Cell formatting includes a variety of features such as font settings (font size, color, character style, etc.), text alignment, background and foreground colors, borders and so on. Formatting options for a cell are defined by the `XICellFormatting` class. This object provides methods and properties that allow you to specify the following types of cell formatting.

- [Predefined Style-Like Formatting](#)
- [Themed Formatting](#)
- [Custom Formatting](#)

To apply the specified format options to a cell, pass the corresponding `XICellFormatting` object to the `IXICell.ApplyFormatting` method as a parameter, or assign it to the `IXICell.Formatting` property.

To format the entire row or column, use the `IXIRow.ApplyFormatting` and `IXIColumn.ApplyFormatting` methods, or `IXIRow.Formatting` and `IXIColumn.Formatting` properties, respectively.

Predefined Style-Like Formatting

Use static properties of the `XICellFormatting` class to format a cell using the predefined format settings corresponding to one of the preset Microsoft® Excel® *styles*. Predefined formatting includes font settings, content alignment, cell borders and fill color. The image below shows cells to which different types of predefined formatting are applied.

	A	B	C	D	E	F
1	Good, Bad and Neutral					
2	Normal	Bad	Good	Neutral		
3						
4	Data and Model					
5	Calculation	Check Cell	Explanatory	Input	Linked Cell	Note
6	Output	Warning Text				
7						
8	Titles and Headings					
9	Heading 1	Heading 2	Heading 3	Heading 4	Title	Total
10						

For an example on how to specify predefined formatting for a cell, refer to the [How to: Apply Predefined Formatting to a Cell](#) topic.

Note that all these static properties return an instance of the `XICellFormatting` class, so that you can use its properties to modify the predefined formatting options to create a [custom format](#).

Themed Formatting

You can use the `XICellFormatting.Themed` method to apply the predefined formatting to a cell that is based on the **document theme**. A document theme is a set of fonts, colors, and graphic effects you can apply to a workbook. Currently, the **Excel Export Library** supports only the *Office* theme (the supported document themes are listed by the `XIDocumentTheme` enumeration). To set the document theme for a workbook, use the `IXIDocument.Theme` property. By default, the *Office 2013* theme is used.

Theme formatting uses six accent colors listed by the `XIThemeColor` enumeration to fill the cell background. The *tint* parameter of the `XICellFormatting.Themed` method allows you to lighten or darken the original theme color. If you apply theme formatting to a cell, the cell font will be automatically set to the body font of the current document theme. The font color depends on the cell background color: if the background color is dark or saturated (the *tint* value is less than **0.5**), the font of the `XIThemeColor.Light1` theme color will be used; and if the background color is light and pale (the *tint* value is greater than or equal to **0.5**), the `XIThemeColor.Dark1` font will be used.

	A	B	C	D	E	F
1	Accent1 20%	Accent2 20%	Accent3 20%	Accent4 20%	Accent5 20%	Accent6 20%
2	Accent1 40%	Accent2 40%	Accent3 40%	Accent4 40%	Accent5 40%	Accent6 40%
3	Accent1 60%	Accent2 60%	Accent3 60%	Accent4 60%	Accent5 60%	Accent6 60%
4	Accent1	Accent2	Accent3	Accent4	Accent5	Accent6
5						

For an example on how to specify theme formatting for a cell, refer to the [How to: Apply Themed Formatting to a Cell](#) topic.

Note that the `XICellFormatting.Themed` method returns an instance of the `XICellFormatting` class, so that you can use its properties to modify current theme formatting settings to create a [custom format](#). You can also use theme formatting elements individually. For example, the `XIColor.FromTheme` method allows you to create a theme-based `XIColor` to be applied anywhere in the document, while the `XIFont.BodyFont` and `XIFont.HeadingsFont` methods enable you to specify the theme body and heading fonts for cell content.

Custom Formatting

The aforementioned methods format a worksheet cell using a predefined set of formatting attributes. However, you may wish to apply your own custom formatting to a cell: color the cell background, adjust font settings, align the cell content, add borders and specify number format options. For this purpose, the `XICellFormatting` object provides a number of properties inherited from the `XIFormatting` class: `XIFormatting.Fill`, `XIFormatting.Font`, `XIFormatting.Alignment`, `XIFormatting.Border` and `XIFormatting.NumberFormat`.

Use these properties to specify the desired formatting settings for a cell, as shown in the example below.

C#

```
// Create a new document and begin to write it to the specified stream.
using (IXlDocument document = exporter.CreateDocument(stream)) {
    // Specify formatting settings to be applied to document content.
    XICellFormatting formatting = new XICellFormatting();
    // Specify cell background color.
    formatting.Fill = XIFill.SolidFill(XIColor.FromArgb(0xCE, 0x8B, 0xDA));
    // Specify font settings (font name, color, size and style).
    formatting.Font = new XIFont();
    formatting.Font.Name = "MV Boli";
    formatting.Font.SchemeStyle = XIFontSchemeStyles.None;
    formatting.Font.Size = 16;
    formatting.Font.Color = Color.Wheat;
    formatting.Font.Bold = true;
    // Specify the alignment of cell content.
    formatting.Alignment = XICellAlignment.FromHV(XIHorizontalAlignment.Center, XIVerticalAlignment.Center);
    // Specify outside border settings.
    formatting.Border = XIBorder.OutlineBorders(XIColor.FromArgb(0x47, 0x7B, 0xD1), XIBorderLineStyle.Thick);
    // Create a new worksheet.
    using (IXlSheet sheet = document.CreateSheet()) {
        // Create the first column and set its width.
        using (IXlColumn column = sheet.CreateColumn())
            column.WidthInPixels = 180;
        // Create the first row in the worksheet.
        using (IXlRow row = sheet.CreateRow()){
            // Create a cell and specify its format settings.
            using (IXlCell cell = row.CreateCell())
            {
                cell.Value = "Custom Format";
                cell.ApplyFormatting(formatting);
            }
        }
    }
}
```

Visual Basic

```
' Create a new document and begin to write it to the specified stream.
Using document As IXlDocument = exporter.CreateDocument(stream)
    ' Specify formatting settings to be applied to document content.
    Dim formatting As New XlCellFormatting()
    ' Specify cell background color.
    formatting.Fill = XlFill.SolidFill(XlColor.FromArgb(&HCE, &H8B, &HDA))
    ' Specify font settings (font name, color, size and style).
    formatting.Font = New XlFont()
    formatting.Font.Name = "MV Boli"
    formatting.Font.SchemeStyle = XlFontSchemeStyles.None
    formatting.Font.Size = 16
    formatting.Font.Color = Color.Wheat
    formatting.Font.Bold = True
    ' Specify the alignment of cell content.
    formatting.Alignment = XlCellAlignment.FromHV(XlHorizontalAlignment.Center, XlVerticalAlignment.Center)
    ' Specify outside border settings.
    formatting.Border = XlBorder.OutlineBorders(XlColor.FromArgb(&H47, &H7B, &HD1), XlBorderLineStyle.Thick)
    ' Create a new worksheet.
    Using sheet As IXlSheet = document.CreateSheet()
        ' Create the first column and set its width.
        Using column As IXlColumn = sheet.CreateColumn()
            column.WidthInPixels = 180
        End Using
        ' Create the first row in the worksheet.
        Using row As IXlRow = sheet.CreateRow()
            ' Create a cell and specify its format settings.
            Using cell As IXlCell = row.CreateCell()
                cell.Value = "Custom Format"
                cell.ApplyFormatting(formatting)
            End Using
        End Using
    End Using
End Using
```

Moreover, the `XlCellFormatting` object implements implicit conversion from the `XlBorder`, `XlCellAlignment`, `XlFill`, `XlFont` and `XlNumberFormat` objects. Therefore, you can directly pass an object specifying the required format characteristics to the `IXlCell.ApplyFormatting` method, or assign it to the `IXlCell.Formatting` property without using unnecessary cast operators.

The following examples elaborate on how to apply different format attributes to a worksheet cell.

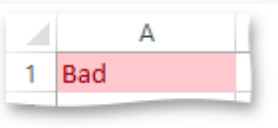
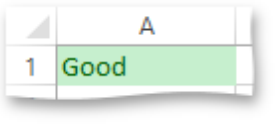
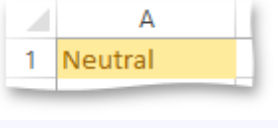
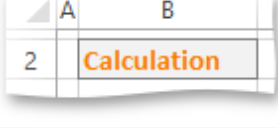
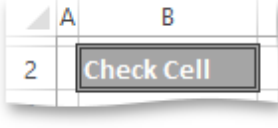
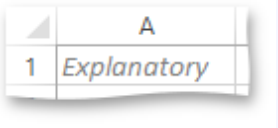
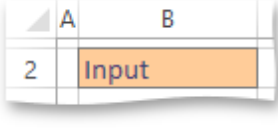
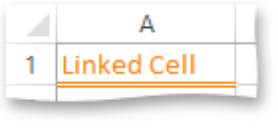
- [How to: Change Cell Background Color](#)
- [How to: Configure Cell Font Settings](#)
- [How to: Add Cell Borders](#)
- [How to: Align Cell Content](#)
- [How to: Specify Number Format for Cell Content](#)

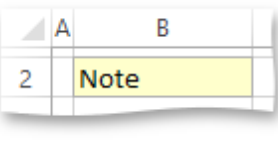
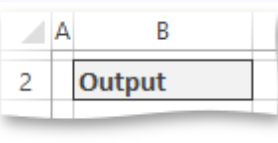
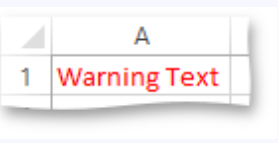
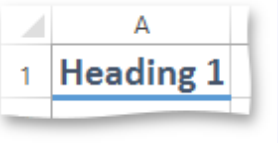
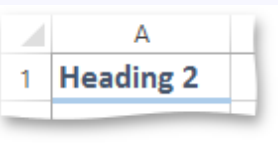
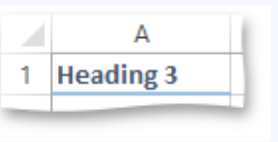
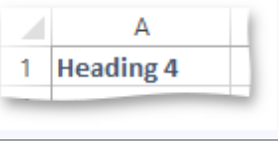
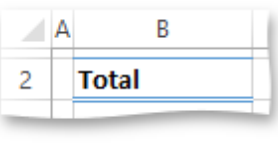
How to: Apply Predefined Formatting to a Cell

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Apply Predefined Formatting to a Cell](#)


Using the **Excel Export** API you can apply a set of predefined format characteristics to a cell in a single step. Predefined formatting corresponds to one of the built-in Microsoft® Excel® styles and includes font settings, alignment options, fill color and cell borders. By default, all cells in a worksheet are formatted in accordance with the *Normal* Excel style.

To change the cell appearance using the predefined format options corresponding to one of the MS Excel styles, use the appropriate static property of the `XICellFormatting` class.

XICellFormatting Property	Excel Built-In Style	Formatted Cell
<code>XICellFormatting.Bad</code>	Bad	
<code>XICellFormatting.Good</code>	Good	
<code>XICellFormatting.Neutral</code>	Neutral	
<code>XICellFormatting.Calculation</code>	Calculation	
<code>XICellFormatting.CheckCell</code>	Check Cell	
<code>XICellFormatting.Explanatory</code>	Explanatory Text	
<code>XICellFormatting.Input</code>	Input	
<code>XICellFormatting.LinkedCell</code>	Linked Cell	

XICellFormatting.Note	Note	
XICellFormatting.Output	Output	
XICellFormatting.WarningText	Warning Text	
XICellFormatting.Heading1	Heading 1	
XICellFormatting.Heading2	Heading 2	
XICellFormatting.Heading3	Heading 3	
XICellFormatting.Heading4	Heading 4	
XICellFormatting.Title	Title	
XICellFormatting.Total	Total	


To apply predefined formatting settings to a cell, pass the XICellFormatting class instance, which is returned by the static property utilized, to the IXICell.ApplyFormatting method as a parameter, or assign it to the IXICell.Formatting property.

 **Tip**

To share formatting settings with multiple cells in a row at once, use the IXIRow.BlankCells and IXIRow.BulkCells methods.

To specify formatting settings for the entire row or column, use the IXIRow.ApplyFormatting and IXIColumn.ApplyFormatting methods, or IXIRow.Formatting and IXIColumn.Formatting properties, respectively.

You can also change the cell appearance using the predefined formatting settings that are based on the **document theme**. For an example, refer to the [How to: Apply Themed Formatting to a Cell](#) document.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(CellFormattingActions.cs)
// Create a new worksheet.
using (IXlSheet sheet = document.CreateSheet()) {
    // Create six successive columns and set their widths.
    for (int i = 0; i < 6; i++) {
        using (IXlColumn column = sheet.CreateColumn()) {
            column.WidthInPixels = 100;
        }
    }
    // Specify the "Good, Bad and Neutral" formatting category.
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Good, Bad and Neutral";
        }
    }
    using (IXlRow row = sheet.CreateRow()) {
        // Create a cell with the default "Normal" formatting.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Normal";
        }
        // Create a cell and apply the "Bad" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Bad";
            cell.Formatting = XlCellFormatting.Bad;
        }
        // Create a cell and apply the "Good" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Good";
            cell.Formatting = XlCellFormatting.Good;
        }
        // Create a cell and apply the "Neutral" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Neutral";
            cell.Formatting = XlCellFormatting.Neutral;
        }
    }
    sheet.SkipRows(1);
    // Specify the "Data and Model" formatting category.
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Data and Model";
        }
    }
    using (IXlRow row = sheet.CreateRow()) {
        // Create a cell and apply the "Calculation" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Calculation";
            cell.Formatting = XlCellFormatting.Calculation;
        }
        // Create a cell and apply the "Check Cell" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Check Cell";
            cell.Formatting = XlCellFormatting.CheckCell;
        }
        // Create a cell and apply the "Explanatory..." predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Explanatory";
            cell.Formatting = XlCellFormatting.Explanatory;
        }
        // Create a cell and apply the "Input" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Input";
            cell.Formatting = XlCellFormatting.Input;
        }
        // Create a cell and apply the "Linked Cell" predefined formatting to it.
```

```

        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Linked Cell";
            cell.Formatting = XlCellFormatting.LinkedCell;
        }
        // Create a cell and apply the "Note" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Note";
            cell.Formatting = XlCellFormatting.Note;
        }
    }
    using (IXlRow row = sheet.CreateRow()) {
        // Create a cell and apply the "Output" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Output";
            cell.Formatting = XlCellFormatting.Output;
        }
        // Create a cell and apply the "Warning Text" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Warning Text";
            cell.Formatting = XlCellFormatting.WarningText;
        }
    }
    sheet.SkipRows(1);
    // Specify the "Titles and Headings" formatting category.
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Titles and Headings";
        }
    }
    using (IXlRow row = sheet.CreateRow()) {
        // Create a cell and apply the "Heading 1" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Heading 1";
            cell.Formatting = XlCellFormatting.Heading1;
        }
        // Create a cell and apply the "Heading 2" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Heading 2";
            cell.Formatting = XlCellFormatting.Heading2;
        }
        // Create a cell and apply the "Heading 3" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Heading 3";
            cell.Formatting = XlCellFormatting.Heading3;
        }
        // Create a cell and apply the "Heading 4" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Heading 4";
            cell.Formatting = XlCellFormatting.Heading4;
        }
        // Create a cell and apply the "Title" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Title";
            cell.Formatting = XlCellFormatting.Title;
        }
        // Create a cell and apply the "Total" predefined formatting to it.
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = "Total";
            cell.Formatting = XlCellFormatting.Total;
        }
    }
}

```

Visual Basic

```
(CellFormattingActions.vb)
' Create a new worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Create six successive columns and set their widths.
    For i As Integer = 0 To 5
        Using column As IXlColumn = sheet.CreateColumn()
            column.WidthInPixels = 100
        End Using
    Next i
    ' Specify the "Good, Bad and Neutral" formatting category.
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Good, Bad and Neutral"
        End Using
    End Using
    Using row As IXlRow = sheet.CreateRow()
        ' Create a cell with the default "Normal" formatting.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Normal"
        End Using
        ' Create a cell and apply the "Bad" predefined formatting to it.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Bad"
            cell.Formatting = XlCellFormatting.Bad
        End Using
        ' Create a cell and apply the "Good" predefined formatting to it.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Good"
            cell.Formatting = XlCellFormatting.Good
        End Using
        ' Create a cell and apply the "Neutral" predefined formatting to it.
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Neutral"
            cell.Formatting = XlCellFormatting.Neutral
        End Using
    End Using
    sheet.SkipRows(1)
    ' Specify the "Data and Model" formatting category.
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Data and Model"
        End Using
    End Using
    Using row As IXlRow = sheet.CreateRow()
        ' Create a cell and apply the "Calculation" predefined formatting
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Calculation"
            cell.Formatting = XlCellFormatting.Calculation
        End Using
        ' Create a cell and apply the "Check Cell" predefined formatting to it.
        Using cell As IXlCell = row.CreateCell()
```

```
        cell.Value = "Check Cell"
        cell.Formatting = XlCellFormatting.CheckCell
    End Using
    ' Create a cell and apply the "Explanatory..." predefined formatting
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Explanatory"
        cell.Formatting = XlCellFormatting.Explanatory
    End Using
    ' Create a cell and apply the "Input" predefined formatting to it.
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Input"
        cell.Formatting = XlCellFormatting.Input
    End Using
    ' Create a cell and apply the "Linked Cell" predefined formatting
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Linked Cell"
        cell.Formatting = XlCellFormatting.LinkedCell
    End Using
    ' Create a cell and apply the "Note" predefined formatting to it.
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Note"
        cell.Formatting = XlCellFormatting.Note
    End Using
End Using
Using row As IXlRow = sheet.CreateRow()
    ' Create a cell and apply the "Output" predefined formatting to it
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Output"
        cell.Formatting = XlCellFormatting.Output
    End Using
    ' Create a cell and apply the "Warning Text" predefined formatting
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Warning Text"
        cell.Formatting = XlCellFormatting.WarningText
    End Using
End Using
sheet.SkipRows(1)
' Specify the "Titles and Headings" formatting category.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Titles and Headings"
    End Using
End Using
Using row As IXlRow = sheet.CreateRow()
    ' Create a cell and apply the "Heading 1" predefined formatting to
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Heading 1"
        cell.Formatting = XlCellFormatting.Heading1
    End Using
    ' Create a cell and apply the "Heading 2" predefined formatting to
    Using cell As IXlCell = row.CreateCell()
```

```
        cell.Value = "Heading 2"
        cell.Formatting = XlCellFormatting.Heading2
    End Using
    ' Create a cell and apply the "Heading 3" predefined formatting to
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Heading 3"
        cell.Formatting = XlCellFormatting.Heading3
    End Using
    ' Create a cell and apply the "Heading 4" predefined formatting to
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Heading 4"
        cell.Formatting = XlCellFormatting.Heading4
    End Using
    ' Create a cell and apply the "Title" predefined formatting to it.
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Title"
        cell.Formatting = XlCellFormatting.Title
    End Using
    ' Create a cell and apply the "Total" predefined formatting to it.
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Total"
        cell.Formatting = XlCellFormatting.Total
    End Using
End Using
End Using
```

See Also[How to: Format a Cell](#)[How to: Apply Themed Formatting to a Cell](#)

How to: Apply Themed Formatting to a Cell

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Apply Themed Formatting to a Cell](#)

Excel Export library provides the capability to change the appearance of worksheet cells using formatting settings that are based on the current **document theme**. A document theme is a set of fonts, colors, and graphic effects you can apply to a workbook. The `XIDocumentTheme` enumeration lists the available document themes (currently, only **Office** themes are supported). Use the `IXIDocument.Theme` property to set the theme of your document. By default, the **Office 2013** theme is used.

To specify themed formatting for a cell, use the `XICellFormatting.Themed` method. Pass the following parameters.

- An `XIThemeColor` enumeration member that specifies one of the six **accent** colors to fill the cell background. These colors are defined by the current document theme and change when another theme is applied to a workbook.
- A **tint** value between **-1.0** and **1.0** used to darken (negative values) or lighten (positive values) the original theme color. **-1.0** means darken 100% (black) and **1.0** means lighten 100% (white). The zero value corresponds to the original color used at full strength.

The `XICellFormatting.Themed` method automatically sets the cell font to the body font of the current document theme (that is *Calibri*, 11 for **Office** themes). The font color depends on the cell background color: if the background color is dark or saturated (the *tint* value is less than **0.5**), the font color will be set to the `XIThemeColor.Light1` theme color; and if the background color is light (the *tint* value is greater than or equal to **0.5**), the `XIThemeColor.Dark1` font color will be used.

To apply theme formatting settings to a cell, pass the `XICellFormatting` object returned by the `XICellFormatting.Themed` method to the `IXICell.ApplyFormatting` method as a parameter, or assign it to the `IXICell.Formatting` property.

Tip

To share formatting settings with multiple cells in a row at once, use the `IXIRow.BlankCells` and `IXIRow.BulkCells` methods. To specify formatting settings for the entire row or column, use the `IXIRow.ApplyFormatting` and `IXIColumn.ApplyFormatting` methods, or `IXIRow.Formatting` and `IXIColumn.Formatting` properties, respectively.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```

(CellFormattingActions.cs)
// Create a worksheet.
using (IXlSheet sheet = document.CreateSheet()) {
    // Create six successive columns and set their widths.
    for (int i = 0; i < 6; i++) {
        using (IXlColumn column = sheet.CreateColumn()) {
            column.WidthInPixels = 100;
        }
    }
    // Specify an array that stores six accent colors of the document theme.
    XlThemeColor[] themeColors = new XlThemeColor[] { XlThemeColor.Accent1, XlThemeColor.Accent2, XlThemeColor.Accent3, XlThemeColor.Accent4, XlThemeColor.Accent5, XlThemeColor.Accent6 };
    // Specify the "20% - AccentN" themed cell formatting.
    // Create a worksheet row.
    using (IXlRow row = sheet.CreateRow()) {
        for (int i = 0; i < 6; i++) {
            // Create a new cell in the row.
            using (IXlCell cell = row.CreateCell()) {
                // Set the cell value.
                cell.Value = string.Format("Accent{0} 20%", i + 1);
                // Apply the themed formatting to the cell using one of the predefined accent colors light
                cell.Formatting = XlCellFormatting.Themed(themeColors[i], 0.8);
            }
        }
    }
    // Specify the "40% - AccentN" themed cell formatting.
    // Create a worksheet row.
    using (IXlRow row = sheet.CreateRow()) {
        for (int i = 0; i < 6; i++) {
            // Create a new cell in the row.
            using (IXlCell cell = row.CreateCell()) {
                // Set the cell value.
                cell.Value = string.Format("Accent{0} 40%", i + 1);
                // Apply the themed formatting to the cell using one of the predefined accent colors light
                cell.Formatting = XlCellFormatting.Themed(themeColors[i], 0.6);
            }
        }
    }
    // Specify the "60% - AccentN" themed cell formatting.
    // Create a worksheet row.
    using (IXlRow row = sheet.CreateRow()) {
        for (int i = 0; i < 6; i++) {
            // Create a new cell in the row.
            using (IXlCell cell = row.CreateCell()) {
                // Set the cell value.
                cell.Value = string.Format("Accent{0} 60%", i + 1);
                // Apply the themed formatting to the cell using one of the predefined accent colors light
                cell.Formatting = XlCellFormatting.Themed(themeColors[i], 0.4);
            }
        }
    }
    // Specify the "AccentN" themed cell formatting.
    // Create a worksheet row.
    using (IXlRow row = sheet.CreateRow()) {
        for (int i = 0; i < 6; i++) {
            // Create a new cell in the row.
            using (IXlCell cell = row.CreateCell()) {
                // Set the cell value.
                cell.Value = string.Format("Accent{0}", i + 1);
                // Apply the themed formatting to the cell using one of the predefined accent colors.
                cell.Formatting = XlCellFormatting.Themed(themeColors[i], 0.0);
            }
        }
    }
}

```

```
(CellFormattingActions.vb)
' Create a worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Create six successive columns and set their widths.
    For i As Integer = 0 To 5
        Using column As IXlColumn = sheet.CreateColumn()
            column.WidthInPixels = 100
        End Using
    Next i
    ' Specify an array that stores six accent colors of the document theme
    Dim themeColors() As XlThemeColor = { XlThemeColor.Accent1, XlThemeCol
    ' Specify the "20% - AccentN" themed cell formatting.
    ' Create a worksheet row.
    Using row As IXlRow = sheet.CreateRow()
        For i As Integer = 0 To 5
            ' Create a new cell in the row.
            Using cell As IXlCell = row.CreateCell()
                ' Set the cell value.
                cell.Value = String.Format("Accent{0} 20%", i + 1)
                ' Apply the themed formatting to the cell using one of the
                cell.Formatting = XlCellFormatting.Themed(themeColors(i),
            End Using
        Next i
    End Using
    ' Specify the "40% - AccentN" themed cell formatting.
    ' Create a worksheet row.
    Using row As IXlRow = sheet.CreateRow()
        For i As Integer = 0 To 5
            ' Create a new cell in the row.
            Using cell As IXlCell = row.CreateCell()
                ' Set the cell value.
                cell.Value = String.Format("Accent{0} 40%", i + 1)
                ' Apply the themed formatting to the cell using one of the
                cell.Formatting = XlCellFormatting.Themed(themeColors(i),
            End Using
        Next i
    End Using
    ' Specify the "60% - AccentN" themed cell formatting.
    ' Create a worksheet row.
    Using row As IXlRow = sheet.CreateRow()
        For i As Integer = 0 To 5
            ' Create a new cell in the row.
            Using cell As IXlCell = row.CreateCell()
                ' Set the cell value.
                cell.Value = String.Format("Accent{0} 60%", i + 1)
                ' Apply the themed formatting to the cell using one of the
                cell.Formatting = XlCellFormatting.Themed(themeColors(i),
            End Using
        Next i
    End Using
    ' Specify the "AccentN" themed cell formatting.
```

```
' Create a worksheet row.
Using row As IXlRow = sheet.CreateRow()
  For i As Integer = 0 To 5
    ' Create a new cell in the row.
    Using cell As IXlCell = row.CreateCell()
      ' Set the cell value.
      cell.Value = String.Format("Accent{0}", i + 1)
      ' Apply the themed formatting to the cell using one of the
      cell.Formatting = XlCellFormatting.Themed(themeColors(i),
    End Using
  Next i
End Using
End Using
```

The images below show the results for different document themes (the workbook is opened in Microsoft® Excel®).

Th
e
m
e

Result

Of
fic
e
20
07
-
20
10

	A	B	C	D	E	F
1	Accent1 20%	Accent2 20%	Accent3 20%	Accent4 20%	Accent5 20%	Accent6 20%
2	Accent1 40%	Accent2 40%	Accent3 40%	Accent4 40%	Accent5 40%	Accent6 40%
3	Accent1 60%	Accent2 60%	Accent3 60%	Accent4 60%	Accent5 60%	Accent6 60%
4	Accent1	Accent2	Accent3	Accent4	Accent5	Accent6
5						

Of
fic
e
20
13
(d
ef
au
lt)

	A	B	C	D	E	F
1	Accent1 20%	Accent2 20%	Accent3 20%	Accent4 20%	Accent5 20%	Accent6 20%
2	Accent1 40%	Accent2 40%	Accent3 40%	Accent4 40%	Accent5 40%	Accent6 40%
3	Accent1 60%	Accent2 60%	Accent3 60%	Accent4 60%	Accent5 60%	Accent6 60%
4	Accent1	Accent2	Accent3	Accent4	Accent5	Accent6
5						

See Also
[How to: Format a Cell](#)
[How to: Apply Predefined Formatting to a Cell](#)

How to: Change Cell Background Color

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Change Cell Background Color](#)

This example demonstrates how to fill a cell background. To do this, use the static methods of the XIFill object.

• **Solid Fill**

To specify the cell background color, call the XIFill.SolidFill method and pass the desired color (defined by the XIColor object) as a parameter. You can use a system-defined color specified by the corresponding property of the [Color](#) structure (in this case, the **Color** object will be converted to the XIColor object using the implicit converter), create a color from the RGB component values (XIColor.FromArgb), or utilize a theme color (XIColor.FromTheme).


• **Pattern Fill**

To specify a pattern style for a cell, use the XIFill.PatternFill method and pass the following parameters.


- Pattern type - specifies the type of a cell's background pattern. The available pattern types are defined by the XIPatternType enumeration members.
- Background color (optional) - specifies the background color of a cell to which the pattern style is applied.
- Pattern color (optional) - specifies the foreground color of the pattern fill.

If you use the XIFill.PatternFill method overload that specifies only a pattern style, the cell background color will be empty and the pattern color will be set to automatic.

To apply fill settings to a cell, pass the specified **XIFill** object to the IXICell.ApplyFormatting method as a parameter, or assign it to the IXICell.Formatting property.

 **Tip**

To share background settings with multiple cells in a row at once, use the IXIRow.BlankCells and IXIRow.BulkCells methods.
To specify fill settings for the entire row or column, use the IXIRow.ApplyFormatting and IXIColumn.ApplyFormatting methods, or IXIRow.Formatting and IXIColumn.Formatting properties, respectively.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(CellFormattingActions.cs)
// Create a new worksheet.
using (IXlSheet sheet = document.CreateSheet()) {
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            // Fill the cell background using the predefined color.
            cell.ApplyFormatting(XlFill.SolidFill(Color.Beige));
        }
        using (IXlCell cell = row.CreateCell()) {
            // Fill the cell background using the custom RGB color.
            cell.ApplyFormatting(XlFill.SolidFill(Color.FromArgb(0xff, 0x99, 0x66)));
        }
        using (IXlCell cell = row.CreateCell()) {
            // Fill the cell background using the theme color.
            cell.ApplyFormatting(XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent3, 0.4)));
        }
    }
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            // Specify the cell background pattern using predefined colors.
            cell.ApplyFormatting(XlFill.PatternFill(XlPatternType.DarkDown, Color.Red, Color.White));
        }
        using (IXlCell cell = row.CreateCell()) {
            // Specify the cell background pattern using custom RGB colors.
            cell.ApplyFormatting(XlFill.PatternFill(XlPatternType.DarkTrellis, Color.FromArgb(0xff, 0xff,
        }
        using (IXlCell cell = row.CreateCell()) {
            // Specify the cell background pattern using theme colors.
            cell.ApplyFormatting(XlFill.PatternFill(XlPatternType.LightHorizontal, XlColor.FromTheme(XlThe
        }
    }
}
```

Visual Basic	
--------------	--

```

(CellFormattingActions.vb)
' Create a new worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            ' Fill the cell background using the predefined color.
            cell.ApplyFormatting(XlFill.SolidFill(Color.Beige))
        End Using
        Using cell As IXlCell = row.CreateCell()
            ' Fill the cell background using the custom RGB color.
            cell.ApplyFormatting(XlFill.SolidFill(Color.FromArgb(&Hff, &H9
        End Using
        Using cell As IXlCell = row.CreateCell()
            ' Fill the cell background using the theme color.
            cell.ApplyFormatting(XlFill.SolidFill(XlColor.FromTheme(XlThem
        End Using
    End Using
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            ' Specify the cell background pattern using predefined colors.
            cell.ApplyFormatting(XlFill.PatternFill(XlPatternType.DarkDown
        End Using
        Using cell As IXlCell = row.CreateCell()
            ' Specify the cell background pattern using custom RGB colors.
            cell.ApplyFormatting(XlFill.PatternFill(XlPatternType.DarkTrel
        End Using
        Using cell As IXlCell = row.CreateCell()
            ' Specify the cell background pattern using theme colors.
            cell.ApplyFormatting(XlFill.PatternFill(XlPatternType.LightHor
        End Using
    End Using
End Using

```

The image below illustrates the result (the workbook is opened in Microsoft® Excel®).

	A	B	C	D
1				
2				
3				

See Also

[How to: Format a Cell](#)

How to: Configure Cell Font Settings

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Configure Cell Font Settings](#)

To set different font attributes for a cell, use the properties of the `XIFont` object. This object provides the following properties inherited from the `XIFontBase` class to change cell font characteristics.

- `XIFontBase.Name` - sets cell font.
- `XIFontBase.Size` - sets cell font size.
- `XIFontBase.Bold` and `XIFontBase.Italic` - sets whether cell font should be bold or italic.
- `XIFontBase.Script` - sets whether cell text should be formatted as superscript or subscript.
- `XIFontBase.StrikeThrough` - sets whether or not cell text should be displayed with a horizontal line through the text.
- `XIFontBase.Underline` - sets the type of underline applied to the font.
- `XIFont.Color` - sets cell font color.
- `XIFont.FontFamily` - sets the font family.

To change font settings of cell content, perform the steps below:

1. Initialize an instance of the `XIFont` class. Do one of the following:
 - Use the `XIFont.XIFont` default constructor.
 - Call the static `XIFont.BodyFont` and `XIFont.HeadingsFont` methods to apply the default theme body and heading fonts to a cell. To set a custom non-theme font of the specified name, size or color, use the `XIFont.CustomFont` method overloads. All these methods return the **XIFont** object, so you can use its properties to specify additional font attributes (font style, underline type, etc.) as described in the next step.
2. Set the required properties of the **XIFont** object. For example, to make cell text bold, set the `XIFontBase.Bold` property to **true**.
3. To apply font characteristics to a cell, use one of the following approaches:
 - Call the `IXICell.ApplyFormatting` method and pass the specified **XIFont** object as a parameter.
 - Assign the specified **XIFont** object to the `IXICell.Formatting` property.

Tip

To share font settings with multiple cells in a row at once, use the `IXIRow.BulkCells` method.

To specify font settings for the entire row or column, use the `IXIRow.ApplyFormatting` and `IXIColumn.ApplyFormatting` methods, or `IXIRow.Formatting` and `IXIColumn.Formatting` properties, respectively.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(CellFormattingActions.cs)
// Create a new worksheet.
using (IXlSheet sheet = document.CreateSheet())
{
    // Create five successive columns and set their widths.
    for (int i = 0; i < 5; i++)
    {
        using (IXlColumn column = sheet.CreateColumn())
        {
            column.WidthInPixels = 100;
        }
    }
    // Create the first row.
    using (IXlRow row = sheet.CreateRow())
    {
        // Create the cell A1.
        using (IXlCell cell = row.CreateCell())
        {
            // Set the cell value.
            cell.Value = "Body font";
            // Apply the theme body font to the cell content.
            cell.ApplyFormatting(XlFont.BodyFont());
        }
        // Create the cell B1.
        using (IXlCell cell = row.CreateCell())
        {
            // Set the cell value.
            cell.Value = "Headings font";
            // Apply the theme heading font to the cell content.
            cell.ApplyFormatting(XlFont.HeadingsFont());
        }
        // Create the cell C1.
        using (IXlCell cell = row.CreateCell())
        {
            // Set the cell value.
            cell.Value = "Custom font";
            // Specify the custom font attributes.
            XlFont font = new XlFont();
            font.Name = "Century Gothic";
            font.SchemeStyle = XlFontSchemeStyles.None;
            // Apply the custom font to the cell content.
            cell.ApplyFormatting(font);
        }
    }
    // Create an array that stores different values of font size.
    int[] fontSizes = new int[] { 11, 14, 18, 24, 36 };
    // Skip one row in the worksheet.
    sheet.SkipRows(1);
    // Create the third row.
    using (IXlRow row = sheet.CreateRow())
    {
        // Create five successive cells (A3:E3) with different font sizes.
        for (int i = 0; i < 5; i++)
        {
            using (IXlCell cell = row.CreateCell())
            {
                // Set the cell value that displays the applied font size.
                cell.Value = string.Format("{0}pt", fontSizes[i]);
                // Create a font instance of the specified size.
                XlFont font = new XlFont();
                font.Size = fontSizes[i];
                // Apply font settings to the cell content.
                cell.ApplyFormatting(font);
            }
        }
    }
}
```

```
}
// Skip one row in the worksheet.
sheet.SkipRows(1);
// Create the fifth row.
using (IXlRow row = sheet.CreateRow())
{
    // Create the cell A5.
    using (IXlCell cell = row.CreateCell())
    {
        // Set the cell value.
        cell.Value = "Red";
        // Create a font instance and set its color.
        XlFont font = new XlFont() { Color = Color.Red };
        // Apply the font color to the cell content.
        cell.ApplyFormatting(font);
    }
    // Create the cell B5.
    using (IXlCell cell = row.CreateCell())
    {
        // Set the cell value.
        cell.Value = "Bold";
        // Create a font instance and set its style to bold.
        XlFont font = new XlFont() { Bold = true };
        // Apply the font style to the cell content.
        cell.ApplyFormatting(font);
    }
    // Create the cell C5.
    using (IXlCell cell = row.CreateCell())
    {
        // Set the cell value.
        cell.Value = "Italic";
        // Create a font instance and set its style to italic.
        XlFont font = new XlFont() { Italic = true };
        // Italicize the cell text.
        cell.ApplyFormatting(font);
    }
    // Create the cell D5.
    using (IXlCell cell = row.CreateCell())
    {
        // Set the cell value.
        cell.Value = "Underline";
        // Create a font instance and set the underline type to double.
        XlFont font = new XlFont() { Underline = XlUnderlineType.Double };
        // Underline the cell text.
        cell.ApplyFormatting(font);
    }
    // Create the cell E5.
    using (IXlCell cell = row.CreateCell())
    {
        // Set the cell value.
        cell.Value = "StrikeThrough";
        // Create a font instance and turn the strikethrough formatting on.
        XlFont font = new XlFont() { StrikeThrough = true };
        // Strike the cell text through.
        cell.ApplyFormatting(font);
    }
}
}
```

Visual Basic

```
(CellFormattingActions.vb)
' Create a new worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Create five successive columns and set their widths.
    For i As Integer = 0 To 4
        Using column As IXlColumn = sheet.CreateColumn()
            column.WidthInPixels = 100
        End Using
    Next i
    ' Create the first row.
    Using row As IXlRow = sheet.CreateRow()
        ' Create the cell A1.
        Using cell As IXlCell = row.CreateCell()
            ' Set the cell value.
            cell.Value = "Body font"
            ' Apply the theme body font to the cell content.
            cell.ApplyFormatting(XlFont.BodyFont())
        End Using
        ' Create the cell B1.
        Using cell As IXlCell = row.CreateCell()
            ' Set the cell value.
            cell.Value = "Headings font"
            ' Apply the theme heading font to the cell content.
            cell.ApplyFormatting(XlFont.HeadingsFont())
        End Using
        ' Create the cell C1.
        Using cell As IXlCell = row.CreateCell()
            ' Set the cell value.
            cell.Value = "Custom font"
            ' Specify the custom font attributes.
            Dim font_Renamed As New XlFont()
            font_Renamed.Name = "Century Gothic"
            font_Renamed.SchemeStyle = XlFontSchemeStyles.None
            ' Apply the custom font to the cell content.
            cell.ApplyFormatting(font_Renamed)
        End Using
    End Using
    ' Create an array that stores different values of font size.
    Dim fontSizes() As Integer = { 11, 14, 18, 24, 36 }
    ' Skip one row in the worksheet.
    sheet.SkipRows(1)
    ' Create the third row.
    Using row As IXlRow = sheet.CreateRow()
        ' Create five successive cells (A3:E3) with different font sizes.
        For i As Integer = 0 To 4
            Using cell As IXlCell = row.CreateCell()
                ' Set the cell value that displays the applied font size.
                cell.Value = String.Format("{0}pt", fontSizes(i))
                ' Create a font instance of the specified size.
                Dim font_Renamed As New XlFont()
                font_Renamed.Size = fontSizes(i)
            End Using
        Next i
    End Using
End Using
```

```
        ' Apply font settings to the cell content.
        cell.ApplyFormatting(font_Renamed)
    End Using
Next i
End Using
' Skip one row in the worksheet.
sheet.SkipRows(1)
' Create the fifth row.
Using row As IXlRow = sheet.CreateRow()
    ' Create the cell A5.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Red"
        ' Create a font instance and set its color.
        Dim font_Renamed As New XlFont() With {.Color = Color.Red}
        ' Apply the font color to the cell content.
        cell.ApplyFormatting(font_Renamed)
    End Using
    ' Create the cell B5.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Bold"
        ' Create a font instance and set its style to bold.
        Dim font_Renamed As New XlFont() With {.Bold = True}
        ' Apply the font style to the cell content.
        cell.ApplyFormatting(font_Renamed)
    End Using
    ' Create the cell C5.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Italic"
        ' Create a font instance and set its style to italic.
        Dim font_Renamed As New XlFont() With {.Italic = True}
        ' Italicize the cell text.
        cell.ApplyFormatting(font_Renamed)
    End Using
    ' Create the cell D5.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Underline"
        ' Create a font instance and set the underline type to double.
        Dim font_Renamed As New XlFont() With {.Underline = XlUnderlin
        ' Underline the cell text.
        cell.ApplyFormatting(font_Renamed)
    End Using
    ' Create the cell E5.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "StrikeThrough"
        ' Create a font instance and turn the strikethrough formatting
        Dim font_Renamed As New XlFont() With {.StrikeThrough = True}
```

```
        ' Strike the cell text through.  
        cell.ApplyFormatting(font_Renamed)  
    End Using  
End Using  
End Using
```

The image below shows the result (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E
1	Body font	Headings font	Custom font		
2					
3	11pt	14pt	18pt	24pt	36pt
4					
5	Red	Bold	Italic	Underline	StrikeThrough
6					

See Also
[How to: Format a Cell](#)

How to: Add Cell Borders

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Add Cell Borders](#)


To specify cell borders, use the `XIBorder` object. This object inherits from the `XIBordersBase` base class, which provides properties used to define border line style options.

To set a particular cell border, create an instance of the `XIBorder` class, and then use the corresponding properties of the `XIBorder` object to specify the border line style and color. The `XIBorderLineStyle` enumerator lists the available line styles. The border color is defined by the `XIColor` value.

You can also use static methods of the `XIBorder` class to set outside borders of a cell at once.


To set	Use API members
A bottom border	<code>XIBordersBase.BottomLineStyle</code> <code>XIBorder.BottomColor</code>
A top border	<code>XIBordersBase.TopLineStyle</code> <code>XIBorder.TopColor</code>
A left border	<code>XIBordersBase.LeftLineStyle</code> <code>XIBorder.LeftColor</code>
A right border	<code>XIBordersBase.RightLineStyle</code> <code>XIBorder.RightColor</code>
Diagonal borders	<code>XIBorder.DiagonalColor</code> <code>XIBordersBase.DiagonalLineStyle</code> <code>XIBordersBase.DiagonalUp</code> <code>XIBordersBase.DiagonalDown</code>
No border	<code>XIBorder.NoBorders</code>
Outside borders	<code>XIBorder.OutlineBorders</code> <code>XIBorder.AllBorders</code>

To apply border settings to a cell, pass the specified **XIBorder** object to the `IXICell.ApplyFormatting` method as a parameter, or assign it to the `IXICell.Formatting` property.

 **Tip**

To share border settings with multiple cells in a row at once, use the `IXIRow.BlankCells` or `IXIRow.BulkCells` method.

To set borders for the entire row or column, use the `IXIRow.ApplyFormatting` and `IXIColumn.ApplyFormatting` methods, or `IXIRow.Formatting` and `IXIColumn.Formatting` properties, respectively.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(CellFormattingActions.cs)
// Specify a two-dimensional array that stores possible line styles for a border.
XlBorderStyle[,] lineStyles = new XlBorderStyle[,] {
    { XlBorderStyle.Thin, XlBorderStyle.Medium, XlBorderStyle.Thick, XlBorderStyle.Dotted, XlBorderStyle.Dashed, XlBorderStyle.DashDot, XlBorderStyle.SlantDashDot, XlBorderStyle.MediumDashed, XlBorderStyle.MediumDashLong, XlBorderStyle.LongDashLong },
    { XlBorderStyle.Dotted, XlBorderStyle.Dashed, XlBorderStyle.DashDot, XlBorderStyle.SlantDashDot, XlBorderStyle.MediumDashed, XlBorderStyle.MediumDashLong, XlBorderStyle.LongDashLong, XlBorderStyle.Thin, XlBorderStyle.Medium, XlBorderStyle.Thick },
    { XlBorderStyle.Thick, XlBorderStyle.Medium, XlBorderStyle.Thin, XlBorderStyle.Dotted, XlBorderStyle.Dashed, XlBorderStyle.DashDot, XlBorderStyle.SlantDashDot, XlBorderStyle.MediumDashed, XlBorderStyle.MediumDashLong, XlBorderStyle.LongDashLong },
    { XlBorderStyle.LongDashLong, XlBorderStyle.MediumDashLong, XlBorderStyle.MediumDashed, XlBorderStyle.SlantDashDot, XlBorderStyle.DashDot, XlBorderStyle.Dashed, XlBorderStyle.Dotted, XlBorderStyle.Thick, XlBorderStyle.Medium, XlBorderStyle.Thin },
    { XlBorderStyle.LongDashLong, XlBorderStyle.MediumDashLong, XlBorderStyle.MediumDashed, XlBorderStyle.SlantDashDot, XlBorderStyle.DashDot, XlBorderStyle.Dashed, XlBorderStyle.Dotted, XlBorderStyle.Thick, XlBorderStyle.Medium, XlBorderStyle.Thin },
    { XlBorderStyle.LongDashLong, XlBorderStyle.MediumDashLong, XlBorderStyle.MediumDashed, XlBorderStyle.SlantDashDot, XlBorderStyle.DashDot, XlBorderStyle.Dashed, XlBorderStyle.Dotted, XlBorderStyle.Thick, XlBorderStyle.Medium, XlBorderStyle.Thin },
    { XlBorderStyle.LongDashLong, XlBorderStyle.MediumDashLong, XlBorderStyle.MediumDashed, XlBorderStyle.SlantDashDot, XlBorderStyle.DashDot, XlBorderStyle.Dashed, XlBorderStyle.Dotted, XlBorderStyle.Thick, XlBorderStyle.Medium, XlBorderStyle.Thin },
    { XlBorderStyle.LongDashLong, XlBorderStyle.MediumDashLong, XlBorderStyle.MediumDashed, XlBorderStyle.SlantDashDot, XlBorderStyle.DashDot, XlBorderStyle.Dashed, XlBorderStyle.Dotted, XlBorderStyle.Thick, XlBorderStyle.Medium, XlBorderStyle.Thin },
    { XlBorderStyle.LongDashLong, XlBorderStyle.MediumDashLong, XlBorderStyle.MediumDashed, XlBorderStyle.SlantDashDot, XlBorderStyle.DashDot, XlBorderStyle.Dashed, XlBorderStyle.Dotted, XlBorderStyle.Thick, XlBorderStyle.Medium, XlBorderStyle.Thin },
    { XlBorderStyle.LongDashLong, XlBorderStyle.MediumDashLong, XlBorderStyle.MediumDashed, XlBorderStyle.SlantDashDot, XlBorderStyle.DashDot, XlBorderStyle.Dashed, XlBorderStyle.Dotted, XlBorderStyle.Thick, XlBorderStyle.Medium, XlBorderStyle.Thin }
};

// Create an exporter instance.
IXlExporter exporter = XlExport.CreateExporter(documentFormat);
// Create a new document.
using (IXlDocument document = exporter.CreateDocument(stream)) {
    document.Options.Culture = CultureInfo.CurrentCulture;
    // Create a worksheet.
    using (IXlSheet sheet = document.CreateSheet()) {
        for (int i = 0; i < 3; i++) {
            sheet.SkipRows(1);
            // Create a worksheet row.
            using (IXlRow row = sheet.CreateRow()) {
                for (int j = 0; j < 4; j++) {
                    row.SkipCells(1);
                    // Create a new cell in the row.
                    using (IXlCell cell = row.CreateCell()) {
                        // Set outside borders for the created cell using a particular line style from the
                        cell.ApplyFormatting(XlBorder.OutlineBorders(Color.SeaGreen, lineStyles[i, j]));
                    }
                }
            }
        }
    }
}
}
```

Visual Basic

```

(CellFormattingActions.vb)
' Specify a two-dimensional array that stores possible line styles for a b
Dim lineStyles(,) As XlBorderLineStyle = { _
    { XlBorderLineStyle.Thin, XlBorderLineStyle.Medium, XlBorderLineStyle.
    { XlBorderLineStyle.Dotted, XlBorderLineStyle.Dashed, XlBorderLineStyle
    { XlBorderLineStyle.SlantDashDot, XlBorderLineStyle.MediumDashed, XlBo
}
' Create an exporter instance.
Dim exporter As IXlExporter = XlExport.CreateExporter(documentFormat)
' Create a new document.
Using document As IXlDocument = exporter.CreateDocument(stream)
    document.Options.Culture = CultureInfo.CurrentCulture
    ' Create a worksheet.
    Using sheet As IXlSheet = document.CreateSheet()
        For i As Integer = 0 To 2
            sheet.SkipRows(1)
            ' Create a worksheet row.
            Using row As IXlRow = sheet.CreateRow()
                For j As Integer = 0 To 3
                    row.SkipCells(1)
                    ' Create a new cell in the row.
                    Using cell As IXlCell = row.CreateCell()
                        ' Set outside borders for the created cell using a
                        cell.ApplyFormatting(XlBorder.OutlineBorders(Color
                    End Using
                Next j
            End Using
        Next i
    End Using
End Using

```

The image below shows the result (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								

See Also

[How to: Format a Cell](#)

How to: Align Cell Content


[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Align Cell Content](#)

To align data contained within a cell, use the corresponding properties of the `XICellAlignment` object.


- `XICellAlignment.HorizontalAlignment`, `XICellAlignment.VerticalAlignment` - set the horizontal and vertical alignment of the cell content.
- `XICellAlignment.Indent` - sets an indent value for text in a cell.
- `XICellAlignment.WrapText` - sets whether or not text should be wrapped in a cell.
- `XICellAlignment.ShrinkToFit` - sets whether or not text should be shrunk to fit the cell width.
- `XICellAlignment.TextRotation` - sets the rotation of the text within a cell.
- `XICellAlignment.ReadingOrder` - sets the reading order for the cell content.

To change alignment characteristics of cell content, perform the steps below.

1. Initialize an instance of the `XICellAlignment` class. Do one of the following.
 - Use the `XICellAlignment.XICellAlignment` default constructor.
 - Call the static `XICellAlignment.FromHV` method to specify both horizontal and vertical alignment for a cell. This method returns the **`XICellAlignment`** object, so that you can use its properties to specify additional alignment options (indent, text rotation, etc.).
2. Set the required properties of the **`XICellAlignment`** object. For example, to wrap long text in a cell into multiple lines, set the `XICellAlignment.WrapText` property to **`true`**.
3. To apply alignment settings to a cell, use one of the following approaches.
 - Call the `IXICell.ApplyFormatting` method and pass the specified **`XICellAlignment`** object as a parameter.
 - Assign the specified **`XICellAlignment`** object to the `IXICell.Formatting` property.

 **Tip**

To share alignment settings with multiple cells in a row at once, use the `IXIRow.BulkCells` method.
To specify alignment characteristics for the entire row or column, use the `IXIRow.ApplyFormatting` and `IXIColumn.ApplyFormatting` methods, or `IXIRow.Formatting` and `IXIColumn.Formatting` properties, respectively.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```

(CellFormattingActions.cs)
// Create a worksheet.
using (IXlSheet sheet = document.CreateSheet()) {
    // Create three successive columns and set their widths.
    for (int i = 0; i < 3; i++) {
        using (IXlColumn column = sheet.CreateColumn()) {
            column.WidthInPixels = 130;
        }
    }
    // Create the first row in the worksheet.
    using (IXlRow row = sheet.CreateRow()) {
        // Set the row height.
        row.HeightInPixels = 40;
        // Create the first cell in the row.
        using (IXlCell cell = row.CreateCell()) {
            // Set the cell value.
            cell.Value = "Left and Top";
            // Specify the horizontal and vertical alignment of the cell content.
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Left, XlVerticalAlignment.Top));
        }
        // Create the second cell in the row.
        using (IXlCell cell = row.CreateCell()) {
            // Set the cell value.
            cell.Value = "Center and Top";
            // Specify the horizontal and vertical alignment of the cell content.
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Center, XlVerticalAlignment.Top));
        }
        // Create the third cell in the row.
        using (IXlCell cell = row.CreateCell()) {
            // Set the cell value.
            cell.Value = "Right and Top";
            // Specify the horizontal and vertical alignment of the cell content.
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Right, XlVerticalAlignment.Top));
        }
    }
    // Create the second row in the worksheet.
    using (IXlRow row = sheet.CreateRow()) {
        // Set the row height.
        row.HeightInPixels = 40;
        // Create the first cell in the row.
        using (IXlCell cell = row.CreateCell()) {
            // Set the cell value.
            cell.Value = "Left and Center";
            // Specify the horizontal and vertical alignment of the cell content.
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Left, XlVerticalAlignment.Center));
        }
        // Create the second cell in the row.
        using (IXlCell cell = row.CreateCell()) {
            // Set the cell value.
            cell.Value = "Center and Center";
            // Specify the horizontal and vertical alignment of the cell content.
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Center, XlVerticalAlignment.Center));
        }
        // Create the third cell in the row.
        using (IXlCell cell = row.CreateCell()) {
            // Set the cell value.
            cell.Value = "Right and Center";
            // Specify the horizontal and vertical alignment of the cell content.
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Right, XlVerticalAlignment.Center));
        }
    }
    // Create the third row in the worksheet.
    using (IXlRow row = sheet.CreateRow()) {
        // Set the row height.
        row.HeightInPixels = 40;
    }
}

```

```

// Create the first cell in the row.
using(IXlCell cell = row.CreateCell()) {
    // Set the cell value.
    cell.Value = "Left and Bottom";
    // Specify the horizontal and vertical alignment of the cell content.
    cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Left, XlVerticalAlignment.Bottom));
}
// Create the second cell in the row.
using(IXlCell cell = row.CreateCell()) {
    // Set the cell value.
    cell.Value = "Center and Bottom";
    // Specify the horizontal and vertical alignment of the cell content.
    cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Center, XlVerticalAlignment.Bottom));
}
// Create the third cell in the row.
using(IXlCell cell = row.CreateCell()) {
    // Set the cell value.
    cell.Value = "Right and Bottom";
    // Specify the horizontal and vertical alignment of the cell content.
    cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Right, XlVerticalAlignment.Bottom));
}
}
sheet.SkipRows(1);
// Create the fifth row in the worksheet.
using(IXlRow row = sheet.CreateRow()) {
    // Create the first cell in the row.
    using(IXlCell cell = row.CreateCell()) {
        // Set the cell value.
        cell.Value = "The WrapText property is applied to wrap the text within a cell";
        // Wrap the text within the cell.
        cell.Formatting = new XlCellAlignment() { WrapText = true };
    }
    // Create the second cell in the row.
    using(IXlCell cell = row.CreateCell()) {
        // Set the cell value.
        cell.Value = "Indented text";
        // Set the indentation of the cell content.
        cell.Formatting = new XlCellAlignment() { Indent = 2 };
    }
    // Create the third cell in the row.
    using(IXlCell cell = row.CreateCell()) {
        // Set the cell value.
        cell.Value = "Rotated text";
        // Rotate the text within the cell.
        cell.Formatting = new XlCellAlignment() { TextRotation = 90 };
    }
}
}

```

Visual Basic

```
(CellFormattingActions.vb)
' Create a worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Create three successive columns and set their widths.
    For i As Integer = 0 To 2
        Using column As IXlColumn = sheet.CreateColumn()
            column.WidthInPixels = 130
        End Using
    Next i
    ' Create the first row in the worksheet.
    Using row As IXlRow = sheet.CreateRow()
        ' Set the row height.
        row.HeightInPixels = 40
        ' Create the first cell in the row.
        Using cell As IXlCell = row.CreateCell()
            ' Set the cell value.
            cell.Value = "Left and Top"
            ' Specify the horizontal and vertical alignment of the cell co
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
        End Using
        ' Create the second cell in the row.
        Using cell As IXlCell = row.CreateCell()
            ' Set the cell value.
            cell.Value = "Center and Top"
            ' Specify the horizontal and vertical alignment of the cell co
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
        End Using
        ' Create the third cell in the row.
        Using cell As IXlCell = row.CreateCell()
            ' Set the cell value.
            cell.Value = "Right and Top"
            ' Specify the horizontal and vertical alignment of the cell co
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
        End Using
    End Using
    ' Create the second row in the worksheet.
    Using row As IXlRow = sheet.CreateRow()
        ' Set the row height.
        row.HeightInPixels = 40
        ' Create the first cell in the row.
        Using cell As IXlCell = row.CreateCell()
            ' Set the cell value.
            cell.Value = "Left and Center"
            ' Specify the horizontal and vertical alignment of the cell co
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
        End Using
        ' Create the second cell in the row.
        Using cell As IXlCell = row.CreateCell()
            ' Set the cell value.
            cell.Value = "Center and Center"
            ' Specify the horizontal and vertical alignment of the cell co
```

```

        cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
End Using
' Create the third cell in the row.
Using cell As IXlCell = row.CreateCell()
    ' Set the cell value.
    cell.Value = "Right and Center"
    ' Specify the horizontal and vertical alignment of the cell co
    cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
End Using
End Using
' Create the third row in the worksheet.
Using row As IXlRow = sheet.CreateRow()
    ' Set the row height.
    row.HeightInPixels = 40
    ' Create the first cell in the row.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Left and Bottom"
        ' Specify the horizontal and vertical alignment of the cell co
        cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
    End Using
    ' Create the second cell in the row.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Center and Bottom"
        ' Specify the horizontal and vertical alignment of the cell co
        cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
    End Using
    ' Create the third cell in the row.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Right and Bottom"
        ' Specify the horizontal and vertical alignment of the cell co
        cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignm
    End Using
End Using
sheet.SkipRows(1)
' Create the fifth row in the worksheet.
Using row As IXlRow = sheet.CreateRow()
    ' Create the first cell in the row.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "The WrapText property is applied to wrap the tex
        ' Wrap the text within the cell.
        cell.Formatting = New XlCellAlignment() With {.WrapText = True
    End Using
    ' Create the second cell in the row.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Indented text"
        ' Set the indentation of the cell content.

```

```
        cell.Formatting = New XlCellAlignment() With {.Indent = 2}
    End Using
    ' Create the third cell in the row.
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Rotated text"
        ' Rotate the text within the cell.
        cell.Formatting = New XlCellAlignment() With {.TextRotation =
    End Using
End Using
End Using
```

The image below shows the result (the workbook is opened in Microsoft® Excel®).

	A	B	C	
1	Left and Top	Center and Top	Right and Top	
2	Left and Center	Center and Center	Right and Center	
3	Left and Bottom	Center and Bottom	Right and Bottom	
4				
5	The WrapText property is applied to wrap the text within a cell	Indented text		Rotated text
6				

See Also
[How to: Format a Cell](#)

How to: Specify Number Format for Cell Content

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Specify Number Format for Cell Content](#)

You can specify how to display a numeric value in a cell by applying *number formats*. For example, a number can appear in a cell as a percentage, decimal, currency, accounting, date or time value. This document provides examples on how to apply different number formats to cell values.

- [Excel Number and Date Formats](#)
- [.NET Number and Date Formats](#)

Excel Number and Date Formats

• Predefined Formats

You can apply one of the preset number or date and time formats to a cell by using static properties of the `XINumberFormat` class. These properties return an object that uses the specific format code to display a numeric value in a cell as a number (`XINumberFormat.Number`, `XINumberFormat.Number2`, `XINumberFormat.NumberWithThousandSeparator`, etc.), percentage (`XINumberFormat.Percentage`, `XINumberFormat.Percentage2`), fraction (`XINumberFormat.Fraction`, `XINumberFormat.Fraction2`), date (`XINumberFormat.ShortDate`, `XINumberFormat.LongDate`, `XINumberFormat.MonthYear`, etc.), time (`XINumberFormat.ShortTime12`, `XINumberFormat.LongTime12`, `XINumberFormat.MinuteSeconds`, etc.), text (`XINumberFormat.Text`) and so on.


• Custom Formats

If the predefined formats do not meet your demands, you can create a custom number format. To do this, assign the corresponding format string to the **NumberFormat** property of the `XICellFormatting` object that defines common formatting settings for a cell. Note that the specified format string will be implicitly converted to the `XINumberFormat` object.

To apply the specified number formatting options to a cell, pass the appropriate `XINumberFormat` or `XICellFormatting` object to the `IXICell.ApplyFormatting` method as a parameter, or assign it to the `IXICell.Formatting` property.

To share number formatting settings with multiple cells in a row at once, use the `IXIRow.BulkCells` method.

To specify a number format for the entire row or column, use the `IXIRow.ApplyFormatting` and `IXIColumn.ApplyFormatting` methods, or `IXIRow.Formatting` and `IXIColumn.Formatting` properties, respectively.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```

(CellFormattingActions.cs)
// Create the header row for the "Excel number formats" category.
using (IXlRow row = sheet.CreateRow()) {
    using (IXlCell cell = row.CreateCell()) {
        // Set the cell value.
        cell.Value = "Excel number formats";
        // Apply the "Heading 4" predefined formatting to the cell.
        cell.Formatting = XlCellFormatting.Heading4;
    }
}
// Use the predefined Excel number formats to display data in cells.
using (IXlRow row = sheet.CreateRow()) {
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Predefined formats:";
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 123.456 as 123.46.
        cell.Value = 123.456;
        cell.Formatting = XlNumberFormat.Number2;
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 12345 as 12,345.
        cell.Value = 12345;
        cell.Formatting = XlNumberFormat.NumberWithThousandSeparator;
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 0.33 as 33%.
        cell.Value = 0.33;
        cell.Formatting = XlNumberFormat.Percentage;
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display the current date as "mm-dd-yy".
        cell.Value = DateTime.Now;
        cell.Formatting = XlNumberFormat.ShortDate;
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display the current time as "h:mm AM/PM".
        cell.Value = DateTime.Now;
        cell.Formatting = XlNumberFormat.ShortTime12;
    }
}
// Use custom number formats to display data in cells.
using (IXlRow row = sheet.CreateRow()) {
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Custom formats:";
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 4310.45 as $4,310.45.
        cell.Value = 4310.45;
        cell.Formatting = new XlCellFormatting();
        cell.Formatting.NumberFormat = @"_([$-409] * #,##0.00_);_([$-409] * \(#,##0.00\);_([$-409] * ""-""";
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 3426.75 as □3,426.75.
        cell.Value = 3426.75;
        cell.Formatting = new XlCellFormatting();
        cell.Formatting.NumberFormat = @"_-$□-2] * #,##0.00_-;-$□-2] * #,##0.00_-;_-$□-2] * "" - ""??_-";
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 0.333 as 33.3%.
        cell.Value = 0.333;
        cell.Formatting = new XlCellFormatting();
        cell.Formatting.NumberFormat = "0.0%";
    }
    using (IXlCell cell = row.CreateCell()) {

```

```
// Apply the custom number format to the date value.  
// Display days as Sunday-Saturday, months as January-December, days as 1-31 and years as 1900-9999  
cell.Value = DateTime.Now;  
cell.Formatting = new XlCellFormatting();  
cell.Formatting.NumberFormat = "dddd, mmmm d, yyyy";  
}  
using (IXlCell cell = row.CreateCell()) {  
    // Display 0.6234 as 341/547.  
    cell.Value = 0.6234;  
    cell.Formatting = new XlCellFormatting();  
    cell.Formatting.NumberFormat = "# ???/???";  
}  
}
```

Visual Basic

```

(CellFormattingActions.vb)
' Create the header row for the "Excel number formats" category.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = "Excel number formats"
        ' Apply the "Heading 4" predefined formatting to the cell.
        cell.Formatting = XlCellFormatting.Heading4
    End Using
End Using
' Use the predefined Excel number formats to display data in cells.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Predefined formats:"
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 123.456 as 123.46.
        cell.Value = 123.456
        cell.Formatting = XlNumberFormat.Number2
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 12345 as 12,345.
        cell.Value = 12345
        cell.Formatting = XlNumberFormat.NumberWithThousandSeparator
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 0.33 as 33%.
        cell.Value = 0.33
        cell.Formatting = XlNumberFormat.Percentage
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display the current date as "mm-dd-yy".
        cell.Value = Date.Now
        cell.Formatting = XlNumberFormat.ShortDate
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display the current time as "h:mm AM/PM".
        cell.Value = Date.Now
        cell.Formatting = XlNumberFormat.ShortTime12
    End Using
End Using
' Use custom number formats to display data in cells.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Custom formats:"
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 4310.45 as $4,310.45.
        cell.Value = 4310.45
        cell.Formatting = New XlCellFormatting()
        cell.Formatting.NumberFormat = "_"([$$-409]* #,##0.00_);_([$$-409]* \(#,##0.00\);_([$$-409]* ""-""
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 3426.75 as □3,426.75.
        cell.Value = 3426.75
        cell.Formatting = New XlCellFormatting()
        cell.Formatting.NumberFormat = "_-[$□-2] * #,##0.00_-;-[$□-2] * #,##0.00_-;-[$□-2] * "" - ""??_-
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 0.333 as 33.3%.
        cell.Value = 0.333
        cell.Formatting = New XlCellFormatting()
        cell.Formatting.NumberFormat = "0.0%"
    End Using
    Using cell As IXlCell = row.CreateCell()

```

```
' Apply the custom number format to the date value.
' Display days as Sunday-Saturday, months as January-December, days as 1-31 and years as 1900-9999
cell.Value = Date.Now
cell.Formatting = New XlCellFormatting()
cell.Formatting.NumberFormat = "dddd, mmmm d, yyyy"
End Using
Using cell As IXlCell = row.CreateCell()
' Display 0.6234 as 341/547.
cell.Value = 0.6234
cell.Formatting = New XlCellFormatting()
cell.Formatting.NumberFormat = "# ???/???"
End Using
End Using
```

.NET Number and Date Formats

To specify the .NET numeric format for a cell value, use the static `XlCellFormatting.FromNetFormat` method with the following parameters:

- A string value that specifies the format string used to format a numeric value in a cell. You can use both standard and custom numeric or date and time format strings.
- A boolean value indicating whether the passed format string is a date and time format string. This parameter is required to distinguish between the standard number and date and time format specifiers. For example, the "d" format specifier can be interpreted as the decimal format specifier or as the short date format specifier depending on the *isDateTimeFormat* parameter value.

You can also use the `XlFormatting.NetFormatString` and `XlFormatting.IsDateTimeFormatString` properties of the `XlCellFormatting` object to specify number formatting options. Note that the `XlCellFormatting.FromNetFormat` method automatically sets these properties according to the values of its parameters.

To apply the specified number formatting options to a cell, pass the `XlCellFormatting` object to the `IXlCell.ApplyFormatting` method as a parameter, or assign it to the `IXlCell.Formatting` property.

To share number formatting settings with multiple cells in a row at once, use the `IXlRow.BulkCells` method.

To specify a number format for the entire row or column, use the `IXlRow.ApplyFormatting` and `IXlColumn.ApplyFormatting` methods, or `IXlRow.Formatting` and `IXlColumn.Formatting` properties, respectively.

Note

Microsoft® Excel® uses number formats similar, but not identical to those of the .NET Framework. To provide compatibility, the **Excel Export** implicitly converts the specified .NET-style format strings to the native Excel formats. But take special note that if the .NET number format is already specified and you try to override it by the [native Excel number format](#), all your settings will be ignored. To remove .NET number format settings, assign an empty string to the `XlFormatting.NetFormatString` property or set it to **null**.

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=T253492>.

C#	
----	--

```
(CellFormattingActions.cs)
// Create the header row for the ".NET number formats" category.
using (IXlRow row = sheet.CreateRow()) {
    using (IXlCell cell = row.CreateCell()) {
        // Set the cell value.
        cell.Value = ".NET number formats";
        // Apply the "Heading 4" predefined formatting to the cell.
        cell.Formatting = XlCellFormatting.Heading4;
    }
}
// Use the standard .NET-style format strings to display data in cells.
using (IXlRow row = sheet.CreateRow()) {
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Standard formats:";
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 123.45 as 123.
        cell.Value = 123.45;
        cell.Formatting = XlCellFormatting.FromNetFormat("D", false);
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 12345 as 1.234500E+004.
        cell.Value = 12345;
        cell.Formatting = XlCellFormatting.FromNetFormat("E", false);
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 0.33 as 33.00%.
        cell.Value = 0.33;
        cell.Formatting = XlCellFormatting.FromNetFormat("P", false);
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display the current date using the short date pattern.
        cell.Value = DateTime.Now;
        cell.Formatting = XlCellFormatting.FromNetFormat("d", true);
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display the current time using the short time pattern.
        cell.Value = DateTime.Now;
        cell.Formatting = XlCellFormatting.FromNetFormat("t", true);
    }
}
// Use custom format strings to display data in cells.
using (IXlRow row = sheet.CreateRow()) {
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Custom formats:";
    }
    using (IXlCell cell = row.CreateCell()) {
        // Display 123.456 as 123.46.
        cell.Value = 123.45;
        cell.Formatting = XlCellFormatting.FromNetFormat("#0.00", false);
    }
}
```

```
using (IXlCell cell = row.CreateCell()) {  
    // Display 12345 as 1.235E+04.  
    cell.Value = 12345;  
    cell.Formatting = XlCellFormatting.FromNetFormat("0.0##e+00", false)  
}  
using (IXlCell cell = row.CreateCell()) {  
    // Display 0.333 as Max=33.3%.  
    cell.Value = 0.333;  
    cell.Formatting = XlCellFormatting.FromNetFormat("Max={0:#.0%}", false)  
}  
using (IXlCell cell = row.CreateCell()) {  
    // Apply the custom format string to the current date.  
    // Display days as 01-31, months as 01-12 and years as a four-digit  
    cell.Value = DateTime.Now;  
    cell.Formatting = XlCellFormatting.FromNetFormat("dd-MM-yyyy", true)  
}  
using (IXlCell cell = row.CreateCell()) {  
    // Apply the custom format string to the current time.  
    // Display hours as 01-12, minutes as 00-59, and add the AM/PM designator  
    cell.Value = DateTime.Now;  
    cell.Formatting = XlCellFormatting.FromNetFormat("hh:mm tt", true)  
}  
}
```

Visual Basic

```
(CellFormattingActions.vb)
' Create the header row for the ".NET number formats" category.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        ' Set the cell value.
        cell.Value = ".NET number formats"
        ' Apply the "Heading 4" predefined formatting to the cell.
        cell.Formatting = XlCellFormatting.Heading4
    End Using
End Using
' Use the standard .NET-style format strings to display data in cells.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Standard formats:"
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 123.45 as 123.
        cell.Value = 123.45
        cell.Formatting = XlCellFormatting.FromNetFormat("D", False)
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 12345 as 1.234500E+004.
        cell.Value = 12345
        cell.Formatting = XlCellFormatting.FromNetFormat("E", False)
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 0.33 as 33.00%.
        cell.Value = 0.33
        cell.Formatting = XlCellFormatting.FromNetFormat("P", False)
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display the current date using the short date pattern.
        cell.Value = Date.Now
        cell.Formatting = XlCellFormatting.FromNetFormat("d", True)
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display the current time using the short time pattern.
        cell.Value = Date.Now
        cell.Formatting = XlCellFormatting.FromNetFormat("t", True)
    End Using
End Using
' Use custom format strings to display data in cells.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Custom formats:"
    End Using
    Using cell As IXlCell = row.CreateCell()
        ' Display 123.456 as 123.46.
        cell.Value = 123.45
        cell.Formatting = XlCellFormatting.FromNetFormat("#0.00", False)
    End Using
```

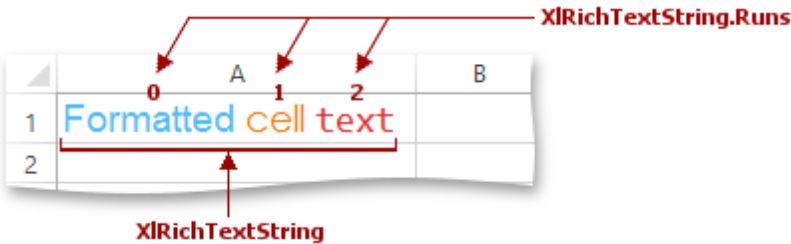
```
Using cell As IXlCell = row.CreateCell()  
    ' Display 12345 as 1.235E+04.  
    cell.Value = 12345  
    cell.Formatting = XlCellFormatting.FromNetFormat("0.0##e+00", Fals  
End Using  
Using cell As IXlCell = row.CreateCell()  
    ' Display 0.333 as Max=33.3%.  
    cell.Value = 0.333  
    cell.Formatting = XlCellFormatting.FromNetFormat("Max={0:#.0%}", F  
End Using  
Using cell As IXlCell = row.CreateCell()  
    ' Apply the custom format string to the current date.  
    ' Display days as 01-31, months as 01-12 and years as a four-digit  
    cell.Value = Date.Now  
    cell.Formatting = XlCellFormatting.FromNetFormat("dd-MM-yyyy", Tru  
End Using  
Using cell As IXlCell = row.CreateCell()  
    ' Apply the custom format string to the current time.  
    ' Display hours as 01-12, minutes as 00-59, and add the AM/PM desi  
    cell.Value = Date.Now  
    cell.Formatting = XlCellFormatting.FromNetFormat("hh:mm tt", True)  
End Using  
End Using
```

See Also[How to: Format a Cell](#)

How to: Apply Rich Formatting to the Cell Text

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Formatting](#) > [How to: Apply Rich Formatting to the Cell Text](#)

The example below demonstrates how to specify the rich formatted text for a cell. Such rich text is defined by the `XIRichTextString` object and includes one or more **text runs** (`XIRichTextRun`). Each run defines a region of the cell text with its own set of font characteristics. All text runs are stored in the collection of runs accessible using the `XIRichTextString.Runs` property. Use the collection's methods to add new runs to the cell text, to modify the existing one, or to delete the specified runs from the collection.



To apply different fonts to specific regions of the cell text, do the following:

1. Initialize a new instance of the `XIRichTextString` class.
2. Specify the required number of text runs that compose the cell text as a whole and add them to collection of runs (`XIRichTextString.Runs`). To create a `XIRichTextRun` object for each individual text region and set its font settings, use the `XIRichTextRun.XIRichTextRun` constructor with the following parameters:
 - A `String` value that specifies the region's content.
 - An `XIFont` object that specifies font settings to be applied to this text region.
3. Call the `IXICell.SetRichText` method and pass the specified `XIRichTextString` object as a parameter.

If you wish to change a region of the cell text after all runs are specified, access the required run by its index in the collection and then use the `XIRichTextRun.Text` and `XIRichTextRun.Font` properties to modify the text and font settings of this run, respectively.

To obtain the entire text contained in a cell, use the `XIRichTextString.Text` property. Note that setting this property to a new value replaces all the existing text runs in the collection with a single run that holds the full text of the cell formatted with the default font.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```

(CellFormattingActions.cs)
// Create a new worksheet.
using (IXlSheet sheet = document.CreateSheet())
{
    // Create the first column and set its width.
    using (IXlColumn column = sheet.CreateColumn())
    {
        column.WidthInPixels = 180;
    }
    // Create the first row.
    using (IXlRow row = sheet.CreateRow())
    {
        // Create the cell A1.
        using (IXlCell cell = row.CreateCell())
        {
            // Create an XlRichTextString instance.
            XlRichTextString richText = new XlRichTextString();
            // Add three text runs to the collection.
            richText.Runs.Add(new XlRichTextRun("Formatted ", XlFont.CustomFont("Arial", 14.0, XlColor.FromArgb(0x000000)));
            richText.Runs.Add(new XlRichTextRun("cell ", XlFont.CustomFont("Century Gothic", 14.0, XlColor.FromArgb(0x000000)));
            richText.Runs.Add(new XlRichTextRun("text", XlFont.CustomFont("Consolas", 14.0, XlColor.FromArgb(0x000000)));
            // Add the rich formatted text to the cell.
            cell.SetRichText(richText);
        }
    }
}

```

Visual Basic

```

(CellFormattingActions.vb)
' Create a new worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Create the first column and set its width.
    Using column As IXlColumn = sheet.CreateColumn()
        column.WidthInPixels = 180
    End Using
    ' Create the first row.
    Using row As IXlRow = sheet.CreateRow()
        ' Create the cell A1.
        Using cell As IXlCell = row.CreateCell()
            ' Create an XlRichTextString instance.
            Dim richText As New XlRichTextString()
            ' Add three text runs to the collection.
            richText.Runs.Add(New XlRichTextRun("Formatted ", XlFont.CustomFont("Arial", 14.0, XlColor.FromArgb(0x000000)))
            richText.Runs.Add(New XlRichTextRun("cell ", XlFont.CustomFont("Century Gothic", 14.0, XlColor.FromArgb(0x000000)))
            richText.Runs.Add(New XlRichTextRun("text", XlFont.CustomFont("Consolas", 14.0, XlColor.FromArgb(0x000000)))
            ' Add the rich formatted text to the cell.
            cell.SetRichText(richText)
        End Using
    End Using
End Using

```

Conditional Formatting

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#)

This section contains the following examples:

- [How to: Format Cell Values that are Above or Below the Average](#)
- [How to: Format Cells that are Less than, Greater than or Between a Value](#)
- [How to: Format Blank Cells](#)
- [How to: Format Unique or Duplicate Values](#)
- [How to: Use a Formula to Determine what Cells to Format](#)
- [How to: Format Top or Bottom Ranked Values](#)
- [How to: Format Cells based on the Text in the Cell](#)
- [How to: Format Cells with Dates](#)
- [How to: Format Cells Using Data Bars](#)
- [How to: Format Cells Using Icon Sets](#)
- [How to: Format Cells Using Color Scales](#)

How to: Format Cell Values that are Above or Below the Average

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cell Values that are Above or Below the Average](#)

- The following example demonstrates how to apply "Above or Below Average" conditional formatting rule to a range of cells.
1. Create new XIConditionalFormatting instance that contains formatting rules and settings.
 2. Specify the range to which the formatting is going to be applied. To do that, add the target range to the object's ranges collection, accessible through the XIConditionalFormatting.Ranges property.
 3. Create new XICondFmtRuleAboveAverage object, representing the new formatting rule.
 4. Set the object's XICondFmtRuleAboveAverage.Condition property to the corresponding XICondFmtAverageCondition enumeration value to specify the formatting condition.
 5. Specify the formatting parameters to the cells, conforming to the condition.
 - To use one of the built-in cell styles, set the **Formatting** property to the corresponding XICellFormatting enumeration value.
 - To apply a custom cell background color, set the XIFormating.Fill property or use one of the XIFill static methods as a **Formatting** property value. The XIFormating.Font property allows you to set the desired font parameters (color, size, etc.)
 6. Add the newly created rule to the corresponding collection contained in the XIConditionalFormatting object by calling the **Add** method.
 7. To activate the created conditional formatting rule, add the object created in step 1 to the to the worksheet collection of conditional formatting rules. The collection can be accessed through the IXISheet.ConditionalFormattings property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```

(ConditionalFormattingActions.cs)
// Create an instance of the XlConditionalFormatting class.
XlConditionalFormatting formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (A1:A11).
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10));
// Create the rule highlighting values that are above the average in the cell range.
XlCondFmtRuleAboveAverage rule = new XlCondFmtRuleAboveAverage();
rule.Condition = XlCondFmtAverageCondition.Above;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Good;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (B1:B11).
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10));
// Create the rule highlighting values that are above or equal to the average value in the cell range.
rule = new XlCondFmtRuleAboveAverage();
rule.Condition = XlCondFmtAverageCondition.AboveOrEqual;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Good;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (C1:C11).
formatting.Ranges.Add(XlCellRange.FromLTRB(2, 0, 2, 10));
// Create the rule highlighting values that are below the average in the cell range.
rule = new XlCondFmtRuleAboveAverage();
rule.Condition = XlCondFmtAverageCondition.Below;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Bad;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (D1:D11).
formatting.Ranges.Add(XlCellRange.FromLTRB(3, 0, 3, 10));
// Create the rule highlighting values that are below or equal to the average value in the cell range.
rule = new XlCondFmtRuleAboveAverage();
rule.Condition = XlCondFmtAverageCondition.BelowOrEqual;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Bad;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);

```

Visual Basic

```
(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10))
' Create the rule highlighting values that are above the average in the cell
Dim rule As New XlCondFmtRuleAboveAverage()
rule.Condition = XlCondFmtAverageCondition.Above
' Specify formatting settings to be applied to cells if the condition is true
rule.Formatting = XlCellFormatting.Good
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional
sheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10))
' Create the rule highlighting values that are above or equal to the average
rule = New XlCondFmtRuleAboveAverage()
rule.Condition = XlCondFmtAverageCondition.AboveOrEqual
' Specify formatting settings to be applied to cells if the condition is true
rule.Formatting = XlCellFormatting.Good
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional
sheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be
formatting.Ranges.Add(XlCellRange.FromLTRB(2, 0, 2, 10))
' Create the rule highlighting values that are below the average in the cell
rule = New XlCondFmtRuleAboveAverage()
rule.Condition = XlCondFmtAverageCondition.Below
' Specify formatting settings to be applied to cells if the condition is true
rule.Formatting = XlCellFormatting.Bad
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional
sheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be
formatting.Ranges.Add(XlCellRange.FromLTRB(3, 0, 3, 10))
' Create the rule highlighting values that are below or equal to the average
rule = New XlCondFmtRuleAboveAverage()
rule.Condition = XlCondFmtAverageCondition.BelowOrEqual
' Specify formatting settings to be applied to cells if the condition is true
rule.Formatting = XlCellFormatting.Bad
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional
sheet.ConditionalFormattings.Add(formatting)
```

As a result, the cells with values less or equal to 6 are be formatted using the "Bad" cell style, and cells with values that are above 6 are formatted using the "Good" cell style.

	A	B	C	D
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4
5	5	5	5	5
6	6	6	6	6
7	7	7	7	7
8	8	8	8	8
9	9	9	9	9
10	10	10	10	10
11	11	11	11	11
12				

How to: Format Cells that are Less than, Greater than or Between a Value

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells that are Less than, Greater than or Between a Value](#)

This example demonstrates how to create the rule that uses a relational operator as a formatting criterion.

1. Create new XIConditionalFormatting instance that contains formatting rules and settings.
2. Specify the range to which the formatting is going to be applied. To do that, add the target range to the object's ranges collection, accessible through the XIConditionalFormatting.Ranges property.
3. Create new XICondFmtRuleCellIs object, representing the new formatting rule.
4. Set the object's XICondFmtRuleCellIs.Operator property to the corresponding XICondFmtOperator enumeration value to specify the formatting condition.
5. To specify the threshold value, use the XICondFmtRuleCellIs.Value property. Note that the value can be represented by a formula.
6. Specify the formatting parameters to the cells, conforming to the condition.
 - To use one of the built-in cell styles, set the **Formatting** property to the corresponding XICellFormatting enumeration value.
 - To apply a custom background color, set the XIFormatting.Fill property or use one of the XIFill static methods as a **Formatting** property value. The XIFormatting.Font property allows you to set the desired font parameters (color, size, etc.)
7. Add the newly created rule to the collection of rules contained in the XIConditionalFormatting object. To do that, use the **Add** method.
8. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the IXISheet.ConditionalFormattings property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XlConditionalFormatting class.
XlConditionalFormatting formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rules should be applied (A1:A11).
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10));
// Create the rule to highlight cells whose values are less than 5.
XlCondFmtRuleCellIs rule = new XlCondFmtRuleCellIs();
rule.Operator = XlCondFmtOperator.LessThan;
rule.Value = 5;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Bad;
formatting.Rules.Add(rule);
// Create the rule to highlight cells whose values are between 5 and 8.
rule = new XlCondFmtRuleCellIs();
rule.Operator = XlCondFmtOperator.Between;
rule.Value = 5;
rule.SecondValue = 8;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Neutral;
formatting.Rules.Add(rule);
// Create the rule to highlight cells whose values are greater than 8.
rule = new XlCondFmtRuleCellIs();
rule.Operator = XlCondFmtOperator.GreaterThan;
rule.Value = 8;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Good;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (B1:B11).
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10));
// Create the rule to highlight cells whose values are greater than a value calculated by a formula.
rule = new XlCondFmtRuleCellIs();
rule.Operator = XlCondFmtOperator.GreaterThan;
rule.Value = "=$A1+3";
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Bad;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```
(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rules should
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10))
' Create the rule to highlight cells whose values are less than 5.
Dim rule As New XlCondFmtRuleCellIs()
rule.Operator = XlCondFmtOperator.LessThan
rule.Value = 5
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Bad
formatting.Rules.Add(rule)
' Create the rule to highlight cells whose values are between 5 and 8.
rule = New XlCondFmtRuleCellIs()
rule.Operator = XlCondFmtOperator.Between
rule.Value = 5
rule.SecondValue = 8
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Neutral
formatting.Rules.Add(rule)
' Create the rule to highlight cells whose values are greater than 8.
rule = New XlCondFmtRuleCellIs()
rule.Operator = XlCondFmtOperator.GreaterThan
rule.Value = 8
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Good
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should b
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10))
' Create the rule to highlight cells whose values are greater than a value
rule = New XlCondFmtRuleCellIs()
rule.Operator = XlCondFmtOperator.GreaterThan
rule.Value = "=$A1+3"
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Bad
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)
```

The image below illustrates the result of code execution.

	A	B	
1	1	12	
2	2	11	
3	3	10	
4	4	9	
5	5	8	
6	6	7	
7	7	6	
8	8	5	
9	9	4	
10	10	3	
11	11	2	
12			

How to: Format Blank Cells

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Blank Cells](#)

The following example demonstrates how to specify the rule that highlights the blank cells.

1. Create new XlConditionalFormatting instance that contains formatting rules and settings.
2. Specify the range to which the formatting is going to be applied by adding it to the ranges collection, accessible through the XlConditionalFormatting.Ranges property.
3. Create new XlCondFmtRuleBlanks object, representing the new formatting rule.
4. Specify the formatting parameters to the cells, conforming to the condition.
 - To use one of the built-in cell styles, set the Formatting property to the corresponding XlCellFormatting enumeration value.
 - To apply a custom background color, use the XlFormatting.Fill property or use one of the XlFill static methods as a Formatting property value. The XlFormatting.Font property allows you to set the desired font parameters (color, size, etc.)
5. Add the newly created rule to the collection of rules contained in the XlConditionalFormatting object. To do that, use the Add method.
6. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the IXlSheet.ConditionalFormattings property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

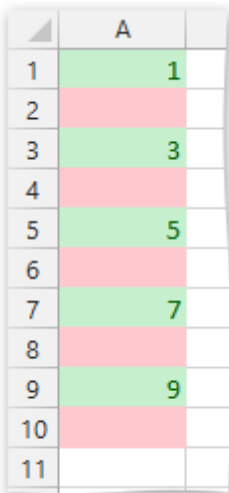
C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XlConditionalFormatting class.
XlConditionalFormatting formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rules should be applied (A1:A10).
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 9));
// Create the rule to highlight blank cells in the range.
XlCondFmtRuleBlanks rule = new XlCondFmtRuleBlanks(true);
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Bad;
formatting.Rules.Add(rule);
// Create the rule to highlight non-blank cells in the range.
rule = new XlCondFmtRuleBlanks(false);
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Good;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```
(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rules should
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 9))
' Create the rule to highlight blank cells in the range.
Dim rule As New XlCondFmtRuleBlanks(True)
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Bad
formatting.Rules.Add(rule)
' Create the rule to highlight non-blank cells in the range.
rule = New XlCondFmtRuleBlanks(False)
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Good
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)
```

The image below illustrates formatted worksheet. The blank cells are formatted using the "Bad" cell style, the non-empty cells are formatted using the "Good" cell style.



	A
1	1
2	
3	3
4	
5	5
6	
7	7
8	
9	9
10	
11	

How to: Format Unique or Duplicate Values

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Unique or Duplicate Values](#)

The following example demonstrates how to specify the rule that identifies duplicate and unique values.

1. Create new XlConditionalFormatting instance that contains formatting rules and settings.
2. Specify the range to which the formatting is going to be applied by adding it to the ranges collection, accessible through the XlConditionalFormatting.Ranges property.
3. Create new XlCondFmtRuleDuplicates object, representing the new formatting rule for duplicate values.
4. Create new XlCondFmtRuleUnique object, representing the new formatting rule for unique values.
5. Specify the formatting parameters to the cells, conforming to the condition.
 - To use one of the built-in cell styles, set the Formatting property to the corresponding XlCellFormatting enumeration value.
 - To apply a custom background color, use the XlFormatting.Fill property or use one of the XlFill static methods as a Formatting property value. The XlFormatting.Font property allows you to set the desired font parameters (color, size, etc.)
6. Add the newly created rule to the collection of rules contained in the XlConditionalFormatting object. To do that, use the Add method.
7. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the IXlSheet.ConditionalFormattings property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XlConditionalFormatting class.
XlConditionalFormatting formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rules should be applied (A1:D11).
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 3, 10));
// Create the rule to identify duplicate values in the cell range.
formatting.Rules.Add(new XlCondFmtRuleDuplicates() { Formatting = XlCellFormatting.Bad });
// Create the rule to identify unique values in the cell range.
formatting.Rules.Add(new XlCondFmtRuleUnique() { Formatting = XlCellFormatting.Good });
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```
(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rules should
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 3, 10))
' Create the rule to identify duplicate values in the cell range.
formatting.Rules.Add(New XlCondFmtRuleDuplicates() With {.Formatting = XlC
' Create the rule to identify unique values in the cell range.
formatting.Rules.Add(New XlCondFmtRuleUnique() With {.Formatting = XlCellF
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)
```

The image below illustrates the result of code execution. The cells with duplicate values are formatted using the "Bad" cell style, the cells with unique values are formatted using the "Good" cell style.

	A	B	C	D
1	1	1	1	1
2	2	3	4	5
3	3	5	7	9
4	4	7	10	13
5	5	9	13	17
6	6	11	16	21
7	7	13	19	25
8	8	15	22	29
9	9	17	25	33
10	10	19	28	37
11	11	21	31	41

How to: Use a Formula to Determine what Cells to Format

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Use a Formula to Determine what Cells to Format](#)

The following example demonstrates how to use a formula as a formatting criterion.

1. Create a new XlConditionalFormatting XlConditionalFormatting instance that contains formatting rules and settings.

2. Specify the cell range to which the formatting is going to be applied. To do that, add the target range to the ranges collection, accessible through the XlConditionalFormatting.Ranges property.

3. Create a new XlCondFmtRuleExpression object with the passed string formula that will be used as a rule criterion.

4. Specify the formatting parameters to cells conforming to the condition.

To use one of the built-in cell styles, set the XlCondFmtRuleWithFormatting.Formatting property to the corresponding XlCellFormatting enumeration value.

To apply a custom background color, use the XlFormatting.Fill property or use one of the XlFill static methods as a **Formatting** property value. The XlFormatting.Font property allows you to set the desired font parameters (color, size, etc.).

5. Add the newly created rule to the corresponding collection contained in the XlConditionalFormatting object by calling the **Add** method.

6. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the IXISheet.ConditionalFormattings property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XlConditionalFormatting class.
XlConditionalFormatting formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rules should be applied (A2:C7).
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 1, 2, 6));
// Create the rule that uses a formula to highlight cells if a value in the column "C" is greater than 0 and less than 50.
XlCondFmtRuleExpression rule = new XlCondFmtRuleExpression("AND($C2>0,$C2<50)");
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlFill.SolidFill(Color.FromArgb(0xff, 0xff, 0xcc));
formatting.Rules.Add(rule);
// Create the rule that uses a formula to highlight cells if a value in the column "C" is less than or equal to 0.
rule = new XlCondFmtRuleExpression("$C2<=0");
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XlCellFormatting.Bad;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```

(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rules should
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 1, 2, 6))
' Create the rule that uses a formula to highlight cells if a value in the
Dim rule As New XlCondFmtRuleExpression("AND($C2>0,$C2<50)")
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlFill.SolidFill(Color.FromArgb(&Hff, &Hff, &Hcc))
formatting.Rules.Add(rule)
' Create the rule that uses a formula to highlight cells if a value in the
rule = New XlCondFmtRuleExpression("$C2<=0")
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Bad
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)

```

The following image demonstrates the result.

	A	B	C	
1	Account ID	User Name	Balance	
2	A105	Berry Dafoe	\$ 155.00	
3	A114	Chris Cadwell	\$ 250.00	
4	B013	Esta Mangold	\$ 48.00	
5	C231	Liam Bell	\$ 350.00	
6	D101	Simon Newman	-\$ 15.00	
7	D105	Wendy Underwood	\$ 10.00	

How to: Format Top or Bottom Ranked Values

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Top or Bottom Ranked Values](#)

The following example demonstrates how to apply a "Top/Bottom N" conditional formatting rule.

1. Create a new `XIConditionalFormatting` instance that contains formatting rules and settings.
2. Specify the cell range to which the formatting is going to be applied. To do that, add the target range to the ranges collection, accessible through the `XIConditionalFormatting.Ranges` property.
3. Create a new `XICondFmtRuleTop10` object that represents the new formatting rule. By default, the created formatting rule will identify top N values. To create a rule to identify bottom N values, set the object's `XICondFmtRuleTop10.Bottom` property to **true**.
4. The default rank value used for the created formatting rule is 10. To change it, specify the `XICondFmtRuleTop10.Rank` property.
5. Specify the formatting parameters to the cells conforming to the condition.
 - To use one of the built-in cell styles, set the `XICondFmtRuleWithFormatting.Formatting` property to the corresponding `XICellFormatting` enumeration value.
 - To apply a custom background color, use the `XIFormatting.Fill` property or use one of the `XIFill` static methods as a `XICondFmtRuleWithFormatting.Formatting` property value. The `XIFormatting.Font` property allows you to set the desired font parameters (color, size, etc.).
6. Add the newly created rule to the corresponding collection contained in the `XIConditionalFormatting` object by calling the **Add** method.
7. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the `IXISheet.ConditionalFormattings` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XIConditionalFormatting class.
XIConditionalFormatting formatting = new XIConditionalFormatting();
// Specify the cell range to which the conditional formatting rules should be applied (A1:D10).
formatting.Ranges.Add(XICellRange.FromLTRB(0, 0, 3, 9));
// Create the rule to identify bottom 10 values in the cell range.
XICondFmtRuleTop10 rule = new XICondFmtRuleTop10();
rule.Bottom = true;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XICellFormatting.Bad;
formatting.Rules.Add(rule);
// Create the rule to identify top 10 values in the cell range.
rule = new XICondFmtRuleTop10();
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XICellFormatting.Good;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```

(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rules should
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 3, 9))
' Create the rule to identify bottom 10 values in the cell range.
Dim rule As New XlCondFmtRuleTop10()
rule.Bottom = True
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Bad
formatting.Rules.Add(rule)
' Create the rule to identify top 10 values in the cell range.
rule = New XlCondFmtRuleTop10()
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Good
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)

```

The image below illustrates the result. The cells with the top 10 values are formatted using the "Good" cell style; top 10 bottom values are formatted using the "Bad" cell style.

	A	B	C	D
1	1	5	9	13
2	2	6	10	14
3	3	7	11	15
4	4	8	12	16
5	5	9	13	17
6	6	10	14	18
7	7	11	15	19
8	8	12	16	20
9	9	13	17	21
10	10	14	18	22

How to: Format Cells based on the Text in the Cell

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells based on the Text in the Cell](#)

The following example demonstrates how to apply a text-based conditional formatting rule.

1. Create a new `XIConditionalFormatting` instance that contains formatting rules and settings.
2. Specify the cell range to which the formatting is going to be applied. To do that, add the target range to the ranges collection, accessible through the `XIConditionalFormatting.Ranges` property.
3. Create a new `XICondFmtRuleSpecificText` object that represents the new formatting rule. Pass the following parameters.
 - One of the `XICondFmtSpecificTextType` enumeration values to specify the formatting rule condition.
 - The string text that will be used as a formatting criterion.
4. Specify the formatting parameters to the cells conforming to the condition.
 - To use one of the built-in cell styles, set the `XICondFmtRuleWithFormatting.Formatting` property to the corresponding `XICellFormatting` enumeration value.
 - To apply a custom background color, use the `XIFormatting.Fill` property or use one of the `XIFill` static methods as a `XICondFmtRuleWithFormatting.Formatting` property value. The `XIFormatting.Font` property allows you to set the desired font parameters (color, size, etc.).
5. Add the newly created rule to the corresponding collection contained in the `XIConditionalFormatting` object by calling the **Add** method.
6. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the `IXISheet.ConditionalFormattings` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XIConditionalFormatting class.
XIConditionalFormatting formatting = new XIConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (B2:B7).
formatting.Ranges.Add(XICellRange.FromLTRB(1, 1, 1, 6));
// Create the rule to highlight cells that contain the given text.
XICondFmtRuleSpecificText rule = new XICondFmtRuleSpecificText(XICondFmtSpecificTextType.Contains, "world");
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XICellFormatting.Neutral;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```
(ConditionalFormattingActions.vb)
' Create an instance of the XIConditionalFormatting class.
Dim formatting As New XIConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be applied (B2:B7).
formatting.Ranges.Add(XICellRange.FromLTRB(1, 1, 1, 6))
' Create the rule to highlight cells that contain the given text.
Dim rule As New XICondFmtRuleSpecificText(XICondFmtSpecificTextType.Contains, "world")
' Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XICellFormatting.Neutral
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting)
```

The following image illustrates the result. The cells with the text that contains the word "worldwide" are formatted using the "Neutral" cell style.

	A	B	C
1	Product	Delivery	Sales
2	Camembert Pierrot	USA	\$ 15,500.00
3	Gorgonzola Telino	Worldwide	\$ 20,250.00
4	Mascarpone Fabioli	USA	\$ 12,634.00
5	Mozzarella di Giovanni	Ships worldwide	\$ 35,010.00
6	Queso Cabrales	Worldwide except EU	\$ 15,234.00
7	Raclette Courdavault	EU	\$ 10,050.00
8			
9			

How to: Format Cells with Dates

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells with Dates](#)

The following example demonstrates how to apply a "A Date Occurring..." formatting rule.

1. Create a new `XIConditionalFormatting` instance that contains formatting rules and settings.
2. Specify the cell range to which the formatting is going to be applied. To do that, add the target range to the ranges collection, accessible through the `XIConditionalFormatting.Ranges` property.
3. Create a new `XICondFmtRuleTimePeriod` object that represents the new formatting rule.
4. Use the object's `XICondFmtRuleTimePeriod.TimePeriod` property to specify the formatting condition by setting it to the corresponding `XICondFmtTimePeriod` enumeration value.
5. Specify the formatting parameters to the cells conforming to the condition.
 - To use one of the built-in cell styles, set the `XICondFmtRuleWithFormatting.Formatting` property to the corresponding `XICellFormatting` enumeration value.
 - To apply a custom background color, use the `XIFormatting.Fill` property or use one of the `XIFill` static methods as a `Formatting` property value. The `XIFormatting.Font` property allows you to set the desired font parameters (color, size, etc.).
6. Add the newly created rule to the corresponding collection of the `XIConditionalFormatting` object by calling the **Add** method.
7. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the `IXISheet.ConditionalFormattings` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XIConditionalFormatting class.
XIConditionalFormatting formatting = new XIConditionalFormatting();
// Specify the cell range to which the conditional formatting rules should be applied (A1:A10).
formatting.Ranges.Add(XICellRange.FromLTRB(0, 0, 0, 9));
// Create the rule to highlight yesterday's dates in the cell range.
XICondFmtRuleTimePeriod rule = new XICondFmtRuleTimePeriod();
rule.TimePeriod = XICondFmtTimePeriod.Yesterday;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XICellFormatting.Bad;
formatting.Rules.Add(rule);
// Create the rule to highlight today's dates in the cell range.
rule = new XICondFmtRuleTimePeriod();
rule.TimePeriod = XICondFmtTimePeriod.Today;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XICellFormatting.Good;
formatting.Rules.Add(rule);
// Create the rule to highlight tomorrows's dates in the cell range.
rule = new XICondFmtRuleTimePeriod();
rule.TimePeriod = XICondFmtTimePeriod.Tomorrow;
// Specify formatting settings to be applied to cells if the condition is true.
rule.Formatting = XICellFormatting.Neutral;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```
(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rules should
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 9))
' Create the rule to highlight yesterday's dates in the cell range.
Dim rule As New XlCondFmtRuleTimePeriod()
rule.TimePeriod = XlCondFmtTimePeriod.Yesterday
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Bad
formatting.Rules.Add(rule)
' Create the rule to highlight today's dates in the cell range.
rule = New XlCondFmtRuleTimePeriod()
rule.TimePeriod = XlCondFmtTimePeriod.Today
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Good
formatting.Rules.Add(rule)
' Create the rule to highlight tomorrows's dates in the cell range.
rule = New XlCondFmtRuleTimePeriod()
rule.TimePeriod = XlCondFmtTimePeriod.Tomorrow
' Specify formatting settings to be applied to cells if the condition is t
rule.Formatting = XlCellFormatting.Neutral
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)
```

The following image illustrates the result.

The cell containing yesterday's date is formatted using the "Bad" cell style, the cell containing today's date is formatted using the "Good" cell style and the cell with tomorrow's date is formatted using the "Neutral" cell style.

	A
1	8/31/2016
2	9/1/2016
3	9/2/2016
4	9/3/2016
5	9/4/2016
6	9/5/2016
7	9/6/2016
8	9/7/2016
9	9/8/2016
10	9/9/2016

How to: Format Cells Using Data Bars

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells Using Data Bars](#)

The following example describes how to apply a data bar conditional formatting rule.

1. Create a new `XIConditionalFormatting` instance that contains formatting rules and settings.
2. Specify the cell range to which the formatting is going to be applied. To do that, add the target range to the ranges collection, accessible through the `XIConditionalFormatting.Ranges` property.
3. Create a new `XICondFmtRuleDataBar` object that represents the new formatting rule.
4. Customize the data bar appearance by specifying its fill color (`XICondFmtRuleDataBar.FillColor`), the color of the borders (`XICondFmtRuleDataBar.BorderColor`) and axis (`XICondFmtRuleDataBar.AxisColor`). To use the gradient fill type, set the `XICondFmtRuleDataBar.GradientFill` to **true**. Setting the `XICondFmtRuleDataBar.MinLength` and `XICondFmtRuleDataBar.MaxLength` properties allow you to specify the minimum and maximum length of the data bar.
5. If necessary, set the minimum and maximum threshold values using the `XICondFmtRuleDataBar.MinValue` and `XICondFmtRuleDataBar.MaxValue` properties.
6. Add the newly created rule to the corresponding collection contained in the `XIConditionalFormatting` object by calling the **Add** method.
7. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the `IXISheet.ConditionalFormattings` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```

(ConditionalFormattingActions.cs)
// Create an instance of the XlConditionalFormatting class.
XlConditionalFormatting formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (A1:A11).
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10));
// Create the rule to compare values in the cell range using data bars.
XlCondFmtRuleDataBar rule = new XlCondFmtRuleDataBar();
// Specify the bar color.
rule.FillColor = XlColor.FromTheme(XlThemeColor.Accent1, 0.2);
// Specify the solid fill type.
rule.GradientFill = false;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (B1:B11).
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10));
// Create the rule to compare values in the cell range using data bars.
rule = new XlCondFmtRuleDataBar();
// Set the positive bar color to green.
rule.FillColor = Color.Green;
// Set the border color of positive bars to green.
rule.BorderColor = Color.Green;
// Set the axis color to brown.
rule.AxisColor = Color.Brown;
// Use the gradient fill type
rule.GradientFill = true;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (C1:C11).
formatting.Ranges.Add(XlCellRange.FromLTRB(2, 0, 2, 10));
// Create the rule to compare values in the cell range using data bars.
rule = new XlCondFmtRuleDataBar();
// Specify the bar color.
rule.FillColor = XlColor.FromTheme(XlThemeColor.Accent4, 0.2);
// Set the minimum length of the data bar.
rule.MinLength = 10;
// Set the maximum length of the data bar.
rule.MaxLength = 90;
// Set the value corresponding to the shortest bar.
rule.MinValue.ObjectType = XlCondFmtValueObjectType.Number;
rule.MinValue.Value = 3;
// Set the direction of data bars.
rule.Direction = XlDataBarDirection.RightToLeft;
// Hide values of cells to which the rule is applied.
rule.ShowValues = false;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);

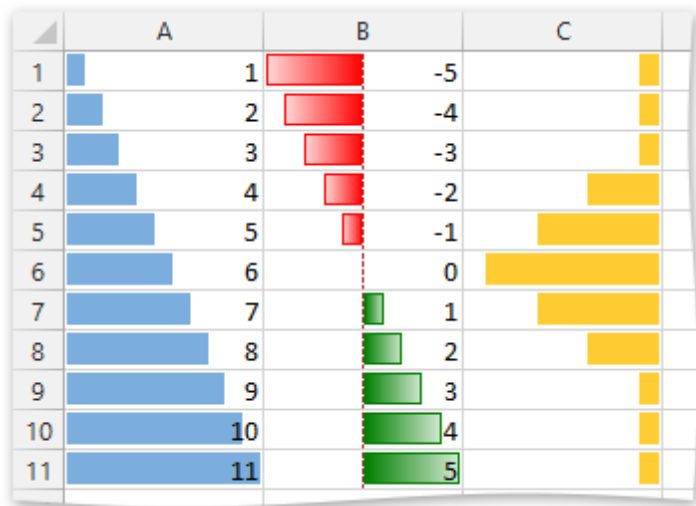
```

Visual Basic

```
(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be applied.
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10))
' Create the rule to compare values in the cell range using data bars.
Dim rule As New XlCondFmtRuleDataBar()
' Specify the bar color.
rule.FillColor = XlColor.FromTheme(XlThemeColor.Accent1, 0.2)
' Specify the solid fill type.
rule.GradientFill = False
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional formatting rules.
worksheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be applied.
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10))
' Create the rule to compare values in the cell range using data bars.
rule = New XlCondFmtRuleDataBar()
' Set the positive bar color to green.
rule.FillColor = Color.Green
' Set the border color of positive bars to green.
rule.BorderColor = Color.Green
' Set the axis color to brown.
rule.AxisColor = Color.Brown
' Use the gradient fill type.
rule.GradientFill = True
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional formatting rules.
worksheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be applied.
formatting.Ranges.Add(XlCellRange.FromLTRB(2, 0, 2, 10))
' Create the rule to compare values in the cell range using data bars.
rule = New XlCondFmtRuleDataBar()
' Specify the bar color.
rule.FillColor = XlColor.FromTheme(XlThemeColor.Accent4, 0.2)
' Set the minimum length of the data bar.
rule.MinLength = 10
' Set the maximum length of the data bar.
rule.MaxLength = 90
' Set the value corresponding to the shortest bar.
rule.MinValue.ObjectType = XlCondFmtValueObjectType.Number
rule.MinValue.Value = 3
' Set the direction of data bars.
rule.Direction = XlDataBarDirection.RightToLeft
' Hide values of cells to which the rule is applied.
rule.ShowValues = False
formatting.Rules.Add(rule)
```

```
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)
```

The image below illustrates the result of the code execution.



How to: Format Cells Using Icon Sets

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells Using Icon Sets](#)

The following example describes how to apply an icon set conditional formatting rule.

1. Create a new `XIConditionalFormatting` instance that contains formatting rules and settings.
2. Specify the cell range to which the formatting is going to be applied. To do that, add the target range to the ranges collection, accessible through the `XIConditionalFormatting.Ranges` property.
3. Create new `XICondFmtRuleIconSet` object that represents the new formatting rule.
4. Specify the icon set that will be used in the formatting rule. You can use one of built-in icon sets as is, or create a custom icon set that contains images from multiple built-in collections.
 - To use one of the built-in icon sets, set the object's `XICondFmtRuleIconSet.IconSetType` property to the corresponding `XICondFmtIconSetType` enumeration value.
 - To create a custom icon set, create custom icons first. To do that, add a new `XICondFmtCustomIcon` object that stores an icon from the built-in set (the required icon set and icon index must be passed to the `XICondFmtCustomIcon` object's constructor method). Then, add your custom icons to the corresponding collection, accessible through the `XICondFmtRuleIconSet.CustomIcons` property. Note that you have to add at least three icons. Otherwise, an exception will be thrown.
5. Add the newly created rule to the corresponding collection contained in the `XIConditionalFormatting` object by calling the **Add** method.
6. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the `IXISheet.ConditionalFormattings` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XlConditionalFormatting class.
XlConditionalFormatting formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (A1:A11).
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10));
// Create the rule to apply a specific icon from the "3 Arrows" icon set to each cell in the range based on
XlCondFmtRuleIconSet rule = new XlCondFmtRuleIconSet();
rule.IconSetType = XlCondFmtIconSetType.Arrows3;
// Set the rule priority.
rule.Priority = 1;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (B1:B11).
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10));
// Create the rule to apply a specific icon from the "3 Flags" icon set to each cell in the range based on
rule = new XlCondFmtRuleIconSet();
rule.IconSetType = XlCondFmtIconSetType.Flags3;
// Set the rule priority.
rule.Priority = 2;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (C1:C11).
formatting.Ranges.Add(XlCellRange.FromLTRB(2, 0, 2, 10));
// Create the rule to apply a specific icon from the "5 Ratings" icon set to each cell in the range based
rule = new XlCondFmtRuleIconSet();
rule.IconSetType = XlCondFmtIconSetType.Rating5;
// Hide values of cells to which the rule is applied.
rule.ShowValues = false;
// Set the rule priority.
rule.Priority = 3;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XlConditionalFormatting class.
formatting = new XlConditionalFormatting();
// Specify the cell range to which the conditional formatting rule should be applied (D1:D11).
formatting.Ranges.Add(XlCellRange.FromLTRB(3, 0, 3, 10));
// Create the rule to apply a specific icon from the "4 Traffic Lights" icon set to each cell in the range
rule = new XlCondFmtRuleIconSet();
rule.IconSetType = XlCondFmtIconSetType.TrafficLights4;
// Reverse the icon order.
rule.Reverse = true;
// Set the rule priority.
rule.Priority = 4;
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```
(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be applied.
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10))
' Create the rule to apply a specific icon from the "3 Arrows" icon set to the range.
Dim rule As New XlCondFmtRuleIconSet()
rule.IconSetType = XlCondFmtIconSetType.Arrows3
' Set the rule priority.
rule.Priority = 1
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional format rules.
sheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be applied.
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10))
' Create the rule to apply a specific icon from the "3 Flags" icon set to the range.
rule = New XlCondFmtRuleIconSet()
rule.IconSetType = XlCondFmtIconSetType.Flags3
' Set the rule priority.
rule.Priority = 2
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional format rules.
sheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be applied.
formatting.Ranges.Add(XlCellRange.FromLTRB(2, 0, 2, 10))
' Create the rule to apply a specific icon from the "5 Ratings" icon set to the range.
rule = New XlCondFmtRuleIconSet()
rule.IconSetType = XlCondFmtIconSetType.Rating5
' Hide values of cells to which the rule is applied.
rule.ShowValues = False
' Set the rule priority.
rule.Priority = 3
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditional format rules.
sheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify the cell range to which the conditional formatting rule should be applied.
formatting.Ranges.Add(XlCellRange.FromLTRB(3, 0, 3, 10))
' Create the rule to apply a specific icon from the "4 Traffic Lights" icon set to the range.
rule = New XlCondFmtRuleIconSet()
rule.IconSetType = XlCondFmtIconSetType.TrafficLights4
' Reverse the icon order.
rule.Reverse = True
' Set the rule priority.
rule.Priority = 4
formatting.Rules.Add(rule)
```

```
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)
```

The image below illustrates the result of the code execution.

	A	B	C	D
1		1	-5	-5
2		2	-4	-4
3		3	-3	-3
4		4	-2	-2
5		5	-1	-1
6		6	0	0
7		7	1	1
8		8	2	2
9		9	3	3
10		10	4	4
11		11	5	5

How to: Format Cells Using Color Scales

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Conditional Formatting](#) > [How to: Format Cells Using Color Scales](#)

This example demonstrates how to create a color scale formatting rule.

1. Create a new `XIConditionalFormatting` instance that contains formatting rules and settings.
2. Specify the cell range to which the formatting is going to be applied. To do that, add the target range to the ranges collection, accessible through the `XIConditionalFormatting.Ranges` property.
3. Create a new `XICondFmtRuleColorScale` object that represents the new formatting rule.
4. By default, the three-scale color rule is created. To create a two-scale color rule, set the object's `XICondFmtRuleColorScale.ColorScaleType` property to the `XICondFmtColorScaleType.ColorScale2` enumeration value.
5. Set a color corresponding to the minimum, midpoint and maximum value in the cell range by specifying the `XICondFmtRuleColorScale.MinColor`, `XICondFmtRuleColorScale.MidpointColor` and `XICondFmtRuleColorScale.MaxColor` properties, respectively.
6. Add the newly created rule to the corresponding collection contained in the `XIConditionalFormatting` object by calling the **Add** method.
7. To activate the created conditional formatting rule, add the object created in step 1 to the worksheet collection of conditional formatting rules. The collection can be accessed through the `IXISheet.ConditionalFormattings` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(ConditionalFormattingActions.cs)
// Create an instance of the XIConditionalFormatting class.
XIConditionalFormatting formatting = new XIConditionalFormatting();
// Specify cell ranges to which the conditional formatting rule should be applied (A1:A11 and C1:C11).
formatting.Ranges.Add(XICellRange.FromLTRB(0, 0, 0, 10));
formatting.Ranges.Add(XICellRange.FromLTRB(2, 0, 2, 10));
// Create the default three-color scale rule to differentiate low, medium and high values in cell ranges.
XICondFmtRuleColorScale rule = new XICondFmtRuleColorScale();
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
// Create an instance of the XIConditionalFormatting class.
formatting = new XIConditionalFormatting();
// Specify cell ranges to which the conditional formatting rule should be applied (B1:B11 and D1:D11).
formatting.Ranges.Add(XICellRange.FromLTRB(1, 0, 1, 10));
formatting.Ranges.Add(XICellRange.FromLTRB(3, 0, 3, 10));
// Create the two-color scale rule to differentiate low and high values in cell ranges.
rule = new XICondFmtRuleColorScale();
rule.ColorScaleType = XICondFmtColorScaleType.ColorScale2;
// Set a color corresponding to the minimum value in the cell range.
rule.MinColor = XIColor.FromTheme(XIThemeColor.Light1, 0.0);
// Set a color corresponding to the maximum value in the cell range.
rule.MaxColor = XIColor.FromTheme(XIThemeColor.Accent1, 0.5);
formatting.Rules.Add(rule);
// Add the specified format options to the worksheet collection of conditional formats.
sheet.ConditionalFormattings.Add(formatting);
```

Visual Basic

```

(ConditionalFormattingActions.vb)
' Create an instance of the XlConditionalFormatting class.
Dim formatting As New XlConditionalFormatting()
' Specify cell ranges to which the conditional formatting rule should be a
formatting.Ranges.Add(XlCellRange.FromLTRB(0, 0, 0, 10))
formatting.Ranges.Add(XlCellRange.FromLTRB(2, 0, 2, 10))
' Create the default three-color scale rule to differentiate low, medium a
Dim rule As New XlCondFmtRuleColorScale()
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)
' Create an instance of the XlConditionalFormatting class.
formatting = New XlConditionalFormatting()
' Specify cell ranges to which the conditional formatting rule should be a
formatting.Ranges.Add(XlCellRange.FromLTRB(1, 0, 1, 10))
formatting.Ranges.Add(XlCellRange.FromLTRB(3, 0, 3, 10))
' Create the two-color scale rule to differentiate low and high values in
rule = New XlCondFmtRuleColorScale()
rule.ColorScaleType = XlCondFmtColorScaleType.ColorScale2
' Set a color corresponding to the minimum value in the cell range.
rule.MinColor = XlColor.FromTheme(XlThemeColor.Light1, 0.0)
' Set a color corresponding to the maximum value in the cell range.
rule.MaxColor = XlColor.FromTheme(XlThemeColor.Accent1, 0.5)
formatting.Rules.Add(rule)
' Add the specified format options to the worksheet collection of conditio
sheet.ConditionalFormattings.Add(formatting)

```

The image below illustrates the result of code execution.

	A	B	C	D
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4
5	5	5	5	5
6	6	6	6	6
7	7	7	7	7
8	8	8	8	8
9	9	9	9	9
10	10	10	10	10
11	11	11	11	11

Tables

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Tables](#)

This section contains the following examples:

- [How to: Create a Table](#)
- [How to: Apply a Table Style](#)
- [How to: Apply Custom Formatting to a Table](#)
- [How to: Create a Calculated Column](#)

How to: Create a Table

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Tables](#) > [How to: Create a Table](#)

This example demonstrates how to use the [Excel Export API](#) to format a cell range containing related data as a table. By using the table features, you can control and manage information in your document more effectively. For example, you can sort and filter table data, use structured references that refer to different table regions in formulas, create calculated columns, display and calculate table totals, and so on.

To organize worksheet data in a table, follow the steps below.

1. To start generating a table, call the `IXIRow.BeginTable` method for the row where you wish for your table to start and pass the following parameters.
- A string array that contains names for the table columns. Column names are displayed in the header row of the table and must be unique within the table. If the specified column names are not unique, a **System.ArgumentException** will be thrown.
 - A Boolean value indicating whether the header row of the table is displayed. If this parameter is **true**, the corresponding cells containing column names will be automatically generated in the header row. If you pass **false**, the header row will be hidden, and therefore you need to add table data to the current row that automatically becomes the first row of the table. Otherwise, an empty row will appear at the beginning of your table.


The third parameter of the `IXIRow.BeginTable` method is optional and allows you to specify formatting settings for the table header row.

2. After you start the table export, you can optionally specify various table settings.
- **Format the table**

You can quickly format the table by applying a built-in table style (`IXITableStyleInfo.Name`) or setting custom formatting for different table regions: the table data range (`IXITable.DataFormatting`) or total row (`IXITable.TotalRowFormatting`).

- **Format individual table columns**

Besides formatting the entire table, you can also apply custom formatting to individual table columns. To do this, access the column you wish to format by its index in the `IXITable.Columns` collection and use one of the following properties of the returned `IXITableColumn` object: the `IXITableColumn.DataFormatting` property - to apply special formatting to the data area of the current column, or the `IXITableColumn.TotalRowFormatting` property - to format the column's total cell.

 **Important**


Note that you should set the required format characteristics for a table or table columns **before** you start generating any table data. Otherwise, you may get a table with partial or incorrect formatting. For more information on how to format a table or its regions, refer to the [How to: Apply Custom Formatting to a Table](#) example.

- **Specify the function to calculate table totals**

You can specify how to calculate a total for each column in the table. To do this, use the column's `IXITableColumn.TotalRowFunction` property that allows you to select one of the predefined functions to calculate the column total. Use the `IXITableColumn.TotalRowLabel` property to specify the text to be displayed in the total row cell of the table column.

3. Generate table rows and populate them with data. This can be done by using regular methods of the **XL Export** library designed for creating worksheet rows and cells (for details, refer to the [How to: Create a Row](#) and [How to: Create a Worksheet Cell and Set Its Value](#) examples). When you add a new row, the table data range expands one row down and table formatting is automatically applied to each row cell that appears within the table data range. When generating table cells, control the accuracy and consistency of table data so that each column in the table contains related data of the same type.
4. After you generate all the required data, finish the table export by calling the `IXIRow.EndTable` method. This method accepts a Boolean parameter specifying whether the table total row is visible. If your table does not require the total row, populate the last row in the table with data and call the row's `IXIRow.EndTable` method with the `hasTotalRow` parameter set to **false**.

After a table is generated, it is automatically added to the read-only `IXITableCollection` collection, which stores all tables in a worksheet and can be accessed by using the `IXISheet.Tables` property.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(TableActions.cs)
IXlTable table;
// Specify an array containing column headings for a table.
string[] columnNames = new string[] { "Product", "Category", "Amount" };
// Create the first row in the worksheet from which the table starts.
using (IXlRow row = sheet.CreateRow())
{
    // Start generating the table with a header row displayed.
    table = row.BeginTable(columnNames, true);
    // Specify the total row label.
    table.Columns[0].TotalRowLabel = "Total";
    // Specify the function to calculate the total.
    table.Columns[2].TotalRowFunction = XlTotalRowFunction.Sum;
    // Specify the number format for the "Amount" column and its total cell.
    XlNumberFormat accounting = @"_([$-409]* #,##0.00_);_([$-409]* \(#,##0.00\);_([$-409]* "-"??_);_";
    table.Columns[2].DataFormatting = accounting;
    table.Columns[2].TotalRowFormatting = accounting;
}
// Generate table rows and populate them with data.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Camembert Pierrot", "Dairy Products", 17000 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Gnocchi di nonna Alice", "Grains/Cereals", 15500 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Mascarpone Fabioli", "Dairy Products", 15000 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Ravioli Angelo", "Grains/Cereals", 12500 }, null);
// Create the total row and finish the table.
using (IXlRow row = sheet.CreateRow())
    row.EndTable(table, true);
```

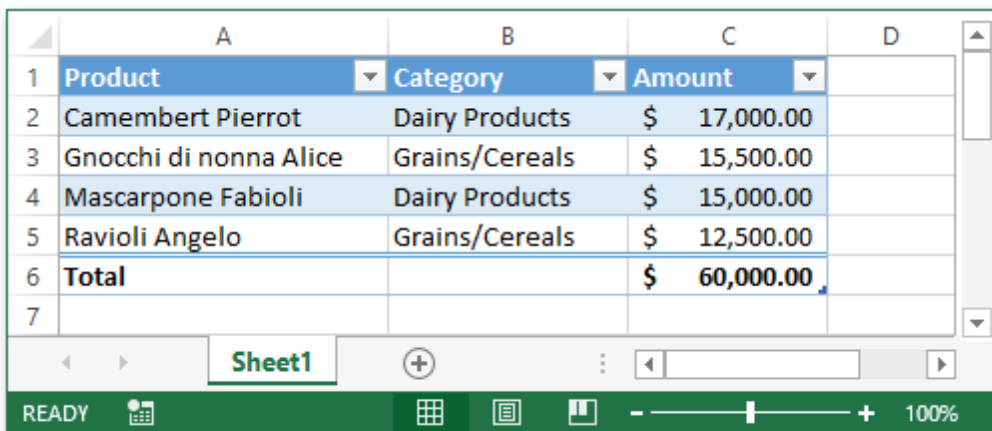
Visual Basic

```

(TableActions.vb)
Dim table As IXlTable
' Specify an array containing column headings for a table.
Dim columnNames() As String = { "Product", "Category", "Amount" }
' Create the first row in the worksheet from which the table starts.
Using row As IXlRow = sheet.CreateRow()
    ' Start generating the table with a header row displayed.
    table = row.BeginTable(columnNames, True)
    ' Specify the total row label.
    table.Columns(0).TotalRowLabel = "Total"
    ' Specify the function to calculate the total.
    table.Columns(2).TotalRowFunction = XlTotalRowFunction.Sum
    ' Specify the number format for the "Amount" column and its total cell
    Dim accounting As XlNumberFormat = "_([$$-409]* #,##0.00_);_([$$-409]*"
    table.Columns(2).DataFormatting = accounting
    table.Columns(2).TotalRowFormatting = accounting
End Using
' Generate table rows and populate them with data.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Camembert Pierrot", "Dairy Products", 17
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Gnocchi di nonna Alice", "Grains/Cereals
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Mascarpone Fabioli", "Dairy Products", 1
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Ravioli Angelo", "Grains/Cereals", 12500
End Using
' Create the total row and finish the table.
Using row As IXlRow = sheet.CreateRow()
    row.EndTable(table, True)
End Using

```

The following image shows the result of the above-mentioned code's execution (the workbook is opened in Microsoft® Excel®).



	A	B	C	D
1	Product	Category	Amount	
2	Camembert Pierrot	Dairy Products	\$ 17,000.00	
3	Gnocchi di nonna Alice	Grains/Cereals	\$ 15,500.00	
4	Mascarpone Fabioli	Dairy Products	\$ 15,000.00	
5	Ravioli Angelo	Grains/Cereals	\$ 12,500.00	
6	Total		\$ 60,000.00	
7				

How to: Apply a Table Style

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Tables](#) > [How to: Apply a Table Style](#)

The **Excel Export** library allows you to format a table by applying one of the Microsoft® Excel® built-in table styles. To do this, use the `IXlTable.Style` property to get access to the `IXlTableStyleInfo` object containing table style options, and then set the `IXlTableStyleInfo.Name` property to the name of the built-in style you wish to apply. The built-in table style names can be obtained as constant fields of the **XlBuiltInTableStyleId** class.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(TableActions.cs)
IXlTable table;
// Specify an array containing column headings for a table.
string[] columnNames = new string[] { "Product", "Category", "Amount" };
// Create the first row in the worksheet from which the table starts.
using (IXlRow row = sheet.CreateRow())
{
    // Start generating the table with a header row displayed.
    table = row.BeginTable(columnNames, true);
    // Apply the table style.
    table.Style.Name = XlBuiltInTableStyleId.Dark7;
}
// Generate table rows and populate them with data.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Camembert Pierrot", "Dairy Products", 17000 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Gnocchi di nonna Alice", "Grains/Cereals", 15500 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Mascarpone Fabioli", "Dairy Products", 15000 }, null);
// Create the last table row and finish the table.
// The total row is not displayed for the table.
using (IXlRow row = sheet.CreateRow())
{
    row.BulkCells(new object[] { "Ravioli Angelo", "Grains/Cereals", 12500 }, null);
    row.EndTable(table, false);
}
```

Visual Basic

```
(TableActions.vb)
Dim table As IXlTable
' Specify an array containing column headings for a table.
Dim columnNames() As String = { "Product", "Category", "Amount" }
' Create the first row in the worksheet from which the table starts.
Using row As IXlRow = sheet.CreateRow()
    ' Start generating the table with a header row displayed.
    table = row.BeginTable(columnNames, True)
    ' Apply the table style.
    table.Style.Name = XlBuiltInTableStyleId.Dark7
End Using
' Generate table rows and populate them with data.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Camembert Pierrot", "Dairy Products", 17000 })
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Gnocchi di nonna Alice", "Grains/Cereals", 15500 })
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Mascarpone Fabioli", "Dairy Products", 15000 })
End Using
' Create the last table row and finish the table.
' The total row is not displayed for the table.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Ravioli Angelo", "Grains/Cereals", 12500 })
    row.EndTable(table, False)
End Using
```

Control Table Style Options

After you apply a table style, you can fine-tune the table appearance by turning style formatting on or off for specific table elements. Table style options are controlled by the corresponding properties of the IXlTableStyleInfo object.

Property	Description															
IXTableStyleInfo.ShowRowStripes	Applies striped row formatting to the table. <table><tr><th>Product</th><th>Category</th><th>Amount</th></tr><tr><td>Camembert Pierrot</td><td>Dairy Products</td><td>17000</td></tr><tr><td>Gnocchi di nonna Alice</td><td>Grains/Cereals</td><td>15500</td></tr><tr><td>Mascarpone Fabioli</td><td>Dairy Products</td><td>15000</td></tr><tr><td>Ravioli Angelo</td><td>Grains/Cereals</td><td>12500</td></tr></table>	Product	Category	Amount	Camembert Pierrot	Dairy Products	17000	Gnocchi di nonna Alice	Grains/Cereals	15500	Mascarpone Fabioli	Dairy Products	15000	Ravioli Angelo	Grains/Cereals	12500
Product	Category	Amount														
Camembert Pierrot	Dairy Products	17000														
Gnocchi di nonna Alice	Grains/Cereals	15500														
Mascarpone Fabioli	Dairy Products	15000														
Ravioli Angelo	Grains/Cereals	12500														
IXTableStyleInfo.ShowColumnStripes	Applies striped column formatting to the table. <table><tr><th>Product</th><th>Category</th><th>Amount</th></tr><tr><td>Camembert Pierrot</td><td>Dairy Products</td><td>17000</td></tr><tr><td>Gnocchi di nonna Alice</td><td>Grains/Cereals</td><td>15500</td></tr><tr><td>Mascarpone Fabioli</td><td>Dairy Products</td><td>15000</td></tr><tr><td>Ravioli Angelo</td><td>Grains/Cereals</td><td>12500</td></tr></table>	Product	Category	Amount	Camembert Pierrot	Dairy Products	17000	Gnocchi di nonna Alice	Grains/Cereals	15500	Mascarpone Fabioli	Dairy Products	15000	Ravioli Angelo	Grains/Cereals	12500
Product	Category	Amount														
Camembert Pierrot	Dairy Products	17000														
Gnocchi di nonna Alice	Grains/Cereals	15500														
Mascarpone Fabioli	Dairy Products	15000														
Ravioli Angelo	Grains/Cereals	12500														

IXITableStyleInfo.ShowFirstColumn Applies style formatting to the first column of the table.

Product	Category	Amount
Camembert Pierrot	Dairy Products	17000
Gnocchi di nonna Alice	Grains/Cereals	15500
Mascarpone Fabioli	Dairy Products	15000
Ravioli Angelo	Grains/Cereals	12500

IXITableStyleInfo.ShowLastColumn Applies style formatting to the last column of the table.

Product	Category	Amount
Camembert Pierrot	Dairy Products	17000
Gnocchi di nonna Alice	Grains/Cereals	15500
Mascarpone Fabioli	Dairy Products	15000
Ravioli Angelo	Grains/Cereals	12500

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#	
----	--

```
(TableActions.cs)
IXlTable table;
// Specify an array containing column headings for tables.
string[] columnNames = new string[] { "Product", "Category", "Amount" };
// Create the row containing the table title.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Disable banded rows" }, XlCellFormatting
sheet.SkipRows(1);
// Start generating the table with a header row displayed.
using (IXlRow row = sheet.CreateRow())
{
    table = row.BeginTable(columnNames, true);
    // Disable banded row formatting for the table.
    table.Style.ShowRowStripes = false;
}
// Generate table rows and populate them with data.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Camembert Pierrot", "Dairy Products", 17
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Gnocchi di nonna Alice", "Grains/Cereals
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Mascarpone Fabioli", "Dairy Products", 1
// Create the last table row and finish the table.
// The total row is not displayed for the table.
using (IXlRow row = sheet.CreateRow())
{
    row.BulkCells(new object[] { "Ravioli Angelo", "Grains/Cereals", 12500
    row.EndTable(table, false);
}
sheet.SkipRows(1);
// Create the row containing the table title.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Enable banded columns" }, XlCellFormatti
sheet.SkipRows(1);
// Start generating the table with a header row displayed.
using (IXlRow row = sheet.CreateRow())
{
    table = row.BeginTable(columnNames, true);
    // Apply banded column formatting to the table.
    table.Style.ShowRowStripes = false;
    table.Style.ShowColumnStripes = true;
}
// Generate table rows and populate them with data.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Camembert Pierrot", "Dairy Products", 17
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Gnocchi di nonna Alice", "Grains/Cereals
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Mascarpone Fabioli", "Dairy Products", 1
// Create the last table row and finish the table.
// The total row is not displayed for the table.
```

```
using (IXlRow row = sheet.CreateRow())
{
    row.BulkCells(new object[] { "Ravioli Angelo", "Grains/Cereals", 12500
    row.EndTable(table, false);
}
sheet.SkipRows(1);
// Create the row containing the table title.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Highlight first column" }, XlCellFormatt
sheet.SkipRows(1);
// Start generating the table with a header row displayed.
using (IXlRow row = sheet.CreateRow())
{
    table = row.BeginTable(columnNames, true);
    // Display special formatting for the first column of the table.
    table.Style.ShowFirstColumn = true;
}
// Generate table rows and populate them with data.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Camembert Pierrot", "Dairy Products", 17
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Gnocchi di nonna Alice", "Grains/Cereals
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Mascarpone Fabioli", "Dairy Products", 1
// Create the last table row and finish the table.
// The total row is not displayed for the table.
using (IXlRow row = sheet.CreateRow())
{
    row.BulkCells(new object[] { "Ravioli Angelo", "Grains/Cereals", 12500
    row.EndTable(table, false);
}
sheet.SkipRows(1);
// Create the row containing the table title.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Highlight last column" }, XlCellFormatti
sheet.SkipRows(1);
// Start generating the table with a header row displayed.
using (IXlRow row = sheet.CreateRow())
{
    table = row.BeginTable(columnNames, true);
    // Display special formatting for the last column of the table.
    table.Style.ShowLastColumn = true;
}
// Generate table rows and populate them with data.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Camembert Pierrot", "Dairy Products", 17
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Gnocchi di nonna Alice", "Grains/Cereals
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Mascarpone Fabioli", "Dairy Products", 1
// Create the last table row and finish the table.
```

```
// The total row is not displayed for the table.
using (IXlRow row = sheet.CreateRow())
{
    row.BulkCells(new object[] { "Ravioli Angelo", "Grains/Cereals", 12500
    row.EndTable(table, false);
}
```

Visual Basic	
--------------	--

```
(TableActions.vb)
Dim table As IXlTable
' Specify an array containing column headings for tables.
Dim columnNames() As String = { "Product", "Category", "Amount" }
' Create the row containing the table title.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Disable banded rows" }, XlCellFormatting)
End Using
sheet.SkipRows(1)
' Start generating the table with a header row displayed.
Using row As IXlRow = sheet.CreateRow()
    table = row.BeginTable(columnNames, True)
    ' Disable banded row formatting for the table.
    table.Style.ShowRowStripes = False
End Using
' Generate table rows and populate them with data.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Camembert Pierrot", "Dairy Products", 17
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Gnocchi di nonna Alice", "Grains/Cereals
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Mascarpone Fabioli", "Dairy Products", 1
End Using
' Create the last table row and finish the table.
' The total row is not displayed for the table.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Ravioli Angelo", "Grains/Cereals", 12500
    row.EndTable(table, False)
End Using
sheet.SkipRows(1)
' Create the row containing the table title.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Enable banded columns" }, XlCellFormatti
End Using
sheet.SkipRows(1)
' Start generating the table with a header row displayed.
Using row As IXlRow = sheet.CreateRow()
    table = row.BeginTable(columnNames, True)
    ' Apply banded column formatting to the table.
    table.Style.ShowRowStripes = False
    table.Style.ShowColumnStripes = True
End Using
' Generate table rows and populate them with data.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Camembert Pierrot", "Dairy Products", 17
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Gnocchi di nonna Alice", "Grains/Cereals
End Using
```

```
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Mascarpone Fabioli", "Dairy Products", 1
End Using
' Create the last table row and finish the table.
' The total row is not displayed for the table.
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Ravioli Angelo", "Grains/Cereals", 12500
    row.EndTable(table, False)
End Using
sheet.SkipRows(1)
' Create the row containing the table title.
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Highlight first column" }, XlCellFormatt
End Using
sheet.SkipRows(1)
' Start generating the table with a header row displayed.
Using row As IXLRow = sheet.CreateRow()
    table = row.BeginTable(columnNames, True)
    ' Display special formatting for the first column of the table.
    table.Style.ShowFirstColumn = True
End Using
' Generate table rows and populate them with data.
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Camembert Pierrot", "Dairy Products", 17
End Using
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Gnocchi di nonna Alice", "Grains/Cereals
End Using
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Mascarpone Fabioli", "Dairy Products", 1
End Using
' Create the last table row and finish the table.
' The total row is not displayed for the table.
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Ravioli Angelo", "Grains/Cereals", 12500
    row.EndTable(table, False)
End Using
sheet.SkipRows(1)
' Create the row containing the table title.
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Highlight last column" }, XlCellFormatti
End Using
sheet.SkipRows(1)
' Start generating the table with a header row displayed.
Using row As IXLRow = sheet.CreateRow()
    table = row.BeginTable(columnNames, True)
    ' Display special formatting for the last column of the table.
    table.Style.ShowLastColumn = True
End Using
' Generate table rows and populate them with data.
Using row As IXLRow = sheet.CreateRow()
```

```
        row.BulkCells(New Object() { "Camembert Pierrot", "Dairy Products", 17
End Using
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Gnocchi di nonna Alice", "Grains/Cereals
End Using
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Mascarpone Fabioli", "Dairy Products", 1
End Using
' Create the last table row and finish the table.
' The total row is not displayed for the table.
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Ravioli Angelo", "Grains/Cereals", 12500
    row.EndTable(table, False)
End Using
```


See Also

[How to: Apply Custom Formatting to a Table](#)

How to: Apply Custom Formatting to a Table

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Tables](#) > [How to: Apply Custom Formatting to a Table](#)

If the [predefined table styles](#) do not comply with your requirements, you can apply custom formatting to different parts of a table. For example, you can fill the background of table cells with the desired color, change cell font attributes, align the cell content, add table borders or specify number format options to be applied to table data.

 **Important**

Due to the infrastructure of the **Excel Export** library, which writes data directly into a stream in consecutive order, you should set the required format characteristics for a table **before** you start generating any table data. Otherwise, you may get a table with partial or incorrect formatting.

Format Table Regions

You can apply specific formatting to different table areas. To format the header row of the table, create the `XICellFormatting` class instance containing the required format characteristics and pass it to the `IXIRow.BeginTable` method as the last parameter. To apply custom formatting to the table's data area or total row, use the `IXITable.DataFormatting` or `IXITable.TotalRowFormatting` property, respectively.


Format Table Columns

If you wish to apply custom formatting to individual table columns, do one of the following.

- Use the `IXIRow.BeginTable` method overload that accepts a list of `XITableColumnInfo` objects as a parameter. An instance of the `XITableColumnInfo` class represents a table column with a specified name and provides a set of properties to format different column regions: the column's header row cell (`XITableColumnInfo.HeaderRowFormatting`), data area (`XITableColumnInfo.DataFormatting`) and total row cell (`XITableColumnInfo.TotalRowFormatting`).

This approach is useful when you need to set special formatting for the header cells of table columns. If it is not necessary, you can customize the appearance of table columns as described below.

- Get access to the column you wish to format by its index in the table's column collection (`IXITable.Columns`) and use one of the following properties of the returned `IXITableColumn` object: the `IXITableColumn.DataFormatting` property - to apply special formatting to the column's data area, or the `IXITableColumn.TotalRowFormatting` property - to format the total cell of the table column.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```

(TableActions.cs)
// Create the first row in the worksheet from which the table starts.
using (IXlRow row = sheet.CreateRow())
{
    XlNumberFormat accounting = @"_([$-409]* #,##0.00_);_([$-409]* \(#,##0.00\);_([$-409]* ""-""??_);_
    // Create objects containing information about table columns (their names and formatting).
    List<XlTableColumnInfo> columns = new List<XlTableColumnInfo>();
    columns.Add(new XlTableColumnInfo("Product"));
    columns.Add(new XlTableColumnInfo("Category"));
    columns.Add(new XlTableColumnInfo("Amount"));
    // Specify formatting settings for the last column of the table.
    columns[2].HeaderRowFormatting = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent6, -0.3));
    columns[2].DataFormatting = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Dark1, 0.9));
    columns[2].DataFormatting.NumberFormat = accounting;
    columns[2].TotalRowFormatting = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Dark1, 0.8));
    columns[2].TotalRowFormatting.NumberFormat = accounting;
    // Specify formatting settings for the header row of the table.
    XlCellFormatting headerRowFormatting = new XlCellFormatting();
    headerRowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent6, 0.0));
    headerRowFormatting.Border = new XlBorder();
    headerRowFormatting.Border.BottomColor = XlColor.FromArgb(0, 0, 0);
    headerRowFormatting.Border.BottomLineStyle = XlBorderLineStyle.Dashed;
    // Start generating the table with a header row displayed.
    IXlTable table = row.BeginTable(columns, true, headerRowFormatting);
    // Apply the table style.
    table.Style.Name = XlBuiltInTableStyleId.Medium16;
    // Disable banded row formatting for the table.
    table.Style.ShowRowStripes = false;
    // Disable the filtering functionality for the table.
    table.HasAutoFilter = false;
    // Specify formatting settings for the total row of the table.
    table.TotalRowFormatting = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Dark1, 0.9));
    table.TotalRowFormatting.Border = new XlBorder()
    {
        BottomColor = XlColor.FromTheme(XlThemeColor.Accent6, 0.0),
        BottomLineStyle = XlBorderLineStyle.Thick,
        TopColor = XlColor.FromArgb(0, 0, 0),
        TopLineStyle = XlBorderLineStyle.Dashed
    };
    // Specify the total row label.
    table.Columns[0].TotalRowLabel = "Total";
    // Specify the function to calculate the total.
    table.Columns[2].TotalRowFunction = XlTotalRowFunction.Sum;
}
// Generate table rows and populate them with data.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Camembert Pierrot", "Dairy Products", 17000 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Gnocchi di nonna Alice", "Grains/Cereals", 15500 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Mascarpone Fabioli", "Dairy Products", 15000 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Ravioli Angelo", "Grains/Cereals", 12500 }, null);
// Create the total row and finish the table.
using (IXlRow row = sheet.CreateRow())
    row.EndTable(sheet.Tables[0], true);

```

Visual Basic

```

(TableActions.vb)
' Create the first row in the worksheet from which the table starts.
Using row As IXLRow = sheet.CreateRow()
    Dim accounting As XLNumberFormat = "_"([$$-409]* #,##0.00_);_([$$-409]*
    ' Create objects containing information about table columns (their nam
    Dim columns As New List(Of XLTableColumnInfo)()
    columns.Add(New XLTableColumnInfo("Product"))
    columns.Add(New XLTableColumnInfo("Category"))
    columns.Add(New XLTableColumnInfo("Amount"))
    ' Specify formatting settings for the last column of the table.
    columns(2).HeaderRowFormatting = XLFill.SolidFill(XLColor.FromTheme(XL
    columns(2).DataFormatting = XLFill.SolidFill(XLColor.FromTheme(XLThemeC
    columns(2).DataFormatting.NumberFormat = accounting
    columns(2).TotalRowFormatting = XLFill.SolidFill(XLColor.FromTheme(XLT
    columns(2).TotalRowFormatting.NumberFormat = accounting
    ' Specify formatting settings for the header row of the table.
    Dim headerRowFormatting As New XLCellFormatting()
    headerRowFormatting.Fill = XLFill.SolidFill(XLColor.FromTheme(XLThemeC
    headerRowFormatting.Border = New XLBorder()
    headerRowFormatting.Border.BottomColor = XLColor.FromArgb(0, 0, 0)
    headerRowFormatting.Border.BottomLineStyle = XLBorderLineStyle.Dashed
    ' Start generating the table with a header row displayed.
    Dim table As IXLTable = row.BeginTable(columns, True, headerRowFormatt
    ' Apply the table style.
    table.Style.Name = XLBuiltInTableStyleId.Medium16
    ' Disable banded row formatting for the table.
    table.Style.ShowRowStripes = False
    ' Disable the filtering functionality for the table.
    table.HasAutoFilter = False
    ' Specify formatting settings for the total row of the table.
    table.TotalRowFormatting = XLFill.SolidFill(XLColor.FromTheme(XLThemeC
    table.TotalRowFormatting.Border = New XLBorder() With {.BottomColor =
    ' Specify the total row label.
    table.Columns(0).TotalRowLabel = "Total"
    ' Specify the function to calculate the total.
    table.Columns(2).TotalRowFunction = XLTotalRowFunction.Sum
End Using
' Generate table rows and populate them with data.
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Camembert Pierrot", "Dairy Products", 17
End Using
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Gnocchi di nonna Alice", "Grains/Cereals
End Using
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Mascarpone Fabioli", "Dairy Products", 1
End Using
Using row As IXLRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Ravioli Angelo", "Grains/Cereals", 12500
End Using
' Create the total row and finish the table.

```

```
Using row As IXLRow = sheet.CreateRow()  
    row.EndTable(sheet.Tables(0), True)  
End Using
```

The following image shows the result of the above-mentioned code's execution (the workbook is opened in Microsoft® Excel®).


The screenshot shows an Excel spreadsheet with a table containing the following data:

	A	B	C	D
1	Product	Category	Amount	
2	Camembert Pierrot	Dairy Products	\$ 17,000.00	
3	Gnocchi di nonna Alice	Grains/Cereals	\$ 15,500.00	
4	Mascarpone Fabioli	Dairy Products	\$ 15,000.00	
5	Ravioli Angelo	Grains/Cereals	\$ 12,500.00	
6	Total		\$ 60,000.00	
7				

How to: Create a Calculated Column

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Tables](#) > [How to: Create a Calculated Column](#)

This example demonstrates how to create a calculated column. The latter uses a single formula that adjusts for each table row. To assign a formula to a table column, use the `IXITableColumn.SetFormula` method. The formula you specify will be automatically filled into all cells of the table column.

 **Important**

Due to the infrastructure of the **Excel Export** library, which writes data directly into a stream in consecutive order, you should assign the required formula to a table column **before** you start generating any table data. Otherwise, a **System.InvalidOperationException** will be thrown.

The `IXITableColumn.SetFormula` method allows you to specify a formula for a calculated column in different ways.

• **Use textual representation of a formula**


You can directly pass a string value representing a formula to be used to calculate column values to the `IXITableColumn.SetFormula` method. A formula string should conform to the formula syntax rules and contain only [supported elements](#) (in particular, you cannot use structured references or defined names in your formula). This is the most common and straightforward way to set a column formula. But to use it, you need to specify a formula parser to provide the capability to parse and validate string formulas. This can be done using the `XIExport.CreateExporter` method with the **DevExpress.Spreadsheet.XIFormulaParser** class instance passed as the parameter. Note that using the **XIFormulaParser** in your code requires a reference to the **DevExpress.Spreadsheet.v18.1.Core.dll** assembly.

• **Compose a formula from tokens**

Use the `IXITableColumn.SetFormula` method with the `XIExpression` parameter to create a formula in a tokenized representation (a.k.a. a *parsed expression*). A parsed expression is a sequence of tokens arranged in Reverse-Polish notation, in which operands are followed by operators. Each token is represented by a descendant of the **DevExpress.Export.XI.XIPtgBase** class. To create a formula of this kind, add the required tokens in the proper order to the `XIExpression` instance and pass it to the `IXITableColumn.SetFormula` method as a parameter. However, this method of creating formulas is too complicated and you will rarely need to use it in your code.

• **Construct a formula from the IXIFormulaParameter objects**

This method enables you to construct a formula for a calculated column from a combination of the most commonly used functions (static methods of the `XIFunc` class), arithmetic and relational operators (static methods of the `XIOper` class) and constants (transformed into the `IXIFormulaParameter` object using the `XIFunc.Param` method). If your formula operates with table data, you can directly refer to specific ranges within the table by using *structured references*. A structured reference is represented by the `XITableReference` class (which also implements the `IXIFormulaParameter` interface) and can be created by using the `IXITable.GetReference` or `IXITable.GetRowReference` method overloads. This approach is demonstrated in the following example that creates a formula to calculate yearly sales for each product in the table and assigns it to the "Yearly Total" column.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(TableActions.cs)
IXlTable table;
// Specify an array containing column headings for a table.
string[] columnNames = new string[] { "Product", "Q1", "Q2", "Q3", "Q4", "Yearly Total" };
// Create the first row in the worksheet from which the table starts.
using (IXlRow row = sheet.CreateRow())
{
    // Start generating the table with a header row displayed.
    table = row.BeginTable(columnNames, true);
    // Specify the total row label.
    table.Columns[0].TotalRowLabel = "Total";
    // Specify the function to calculate the total.
    table.Columns[5].TotalRowFunction = XlTotalRowFunction.Sum;
    // Specify the number format for numeric values in the table and the total cell of the "Yearly Total"
    XlNumberFormat accounting = @"_([$-409]* #,##0.00_);_([$-409]* \(#,##0.00\);_([$-409]* "-"??_);_
    table.DataFormatting = accounting;
    table.Columns[5].TotalRowFormatting = accounting;
    // Set the formula to calculate annual sales of each product
    // and display results in the "Yearly Total" column.
    table.Columns[5].SetFormula(XlFunc.Sum(table.GetRowReference("Q1", "Q4")));
}
// Generate table rows and populate them with data.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Camembert Pierrot", 17000, 18500, 17500, 18000 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Gnocchi di nonna Alice", 15500, 14500, 15000, 14000 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Mascarpone Fabioli", 15000, 15750, 16000, 15500 }, null);
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new object[] { "Ravioli Angelo", 12500, 11000, 13500, 12000 }, null);
// Create the total row and finish the table.
using (IXlRow row = sheet.CreateRow())
    row.EndTable(table, true);
```

Visual Basic

```

(TableActions.vb)
Dim table As IXlTable
' Specify an array containing column headings for a table.
Dim columnNames() As String = { "Product", "Q1", "Q2", "Q3", "Q4", "Yearly Total" }
' Create the first row in the worksheet from which the table starts.
Using row As IXlRow = sheet.CreateRow()
    ' Start generating the table with a header row displayed.
    table = row.BeginTable(columnNames, True)
    ' Specify the total row label.
    table.Columns(0).TotalRowLabel = "Total"
    ' Specify the function to calculate the total.
    table.Columns(5).TotalRowFunction = XlTotalRowFunction.Sum
    ' Specify the number format for numeric values in the table and the total row label.
    Dim accounting As XlNumberFormat = "_([$$-409]* #,##0.00_);_([$$-409]* #,##0.00_)"
    table.DataFormatting = accounting
    table.Columns(5).TotalRowFormatting = accounting
    ' Set the formula to calculate annual sales of each product
    ' and display results in the "Yearly Total" column.
    table.Columns(5).SetFormula(XlFunc.Sum(table.GetRowReference("Q1", "Q4")))
End Using
' Generate table rows and populate them with data.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Camembert Pierrot", 17000, 18500, 17500, 18000 })
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Gnocchi di nonna Alice", 15500, 14500, 15000, 14000 })
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Mascarpone Fabioli", 15000, 15750, 16000, 15500 })
End Using
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New Object() { "Ravioli Angelo", 12500, 11000, 13500, 12000 })
End Using
' Create the total row and finish the table.
Using row As IXlRow = sheet.CreateRow()
    row.EndTable(table, True)
End Using

```

The image below illustrates the result (the workbook is opened in Microsoft® Excel®).

	A	B	C	D	E	F
1	Product	Q1	Q2	Q3	Q4	Yearly Total
2	Camembert Pierrot	\$ 17,000.00	\$ 18,500.00	\$ 17,500.00	\$ 18,000.00	=SUM(Table1[@[Q1]:[Q4]])
3	Gnocchi di nonna Alice	\$ 15,500.00	\$ 14,500.00	\$ 15,000.00	\$ 14,000.00	\$ 59,000.00
4	Mascarpone Fabioli	\$ 15,000.00	\$ 15,750.00	\$ 16,000.00	\$ 15,500.00	\$ 62,250.00
5	Ravioli Angelo	\$ 12,500.00	\$ 11,000.00	\$ 13,500.00	\$ 12,000.00	\$ 49,000.00
6	Total					\$ 241,250.00
7						

Grouping

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Grouping](#)

The following example demonstrates how to group worksheet data.

To group rows or columns, call the IXISheet.BeginGroup method to start grouping, create the necessary number of rows/columns and finalize it by calling the IXISheet.EndGroup method.

To create nested groups, call the IXISheet.BeginGroup - IXISheet.EndGroup paired methods inside another pair, as illustrated in the code snippet below. If necessary, pass the outline level number as the IXISheet.BeginGroup method parameter.

The grouping direction can be specified by setting the IXIOutlineProperties.SummaryBelow and IXIOutlineProperties.SummaryRight properties.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

In this code snippet, one level of grouped columns and two levels of grouped rows are created. After the outer group of rows is created, the IXISheet.BeginGroup method is called again to create the inner group of rows. Note that both methods are enclosed with paired IXISheet.EndGroup methods.

C#

```

(DataActions.cs)
// Create an exporter instance.
IXlExporter exporter = XlExport.CreateExporter(documentFormat);
// Create a new document.
using (IXlDocument document = exporter.CreateDocument(stream)) {
    document.Options.Culture = CultureInfo.CurrentCulture;
    // Create a worksheet.
    using (IXlSheet sheet = document.CreateSheet()) {
        // Specify the summary row and summary column location for the grouped data.
        sheet.OutlineProperties.SummaryBelow = true;
        sheet.OutlineProperties.SummaryRight = true;
        // Create the column "A" and set its width.
        using (IXlColumn column = sheet.CreateColumn()) {
            column.WidthInPixels = 200;
        }
        // Begin to group worksheet columns starting from the column "B" to the column "E".
        sheet.BeginGroup(false);
        // Create four successive columns ("B", "C", "D" and "E") and set the specific number format for
        for (int i = 0; i < 4; i++) {
            using (IXlColumn column = sheet.CreateColumn()) {
                column.WidthInPixels = 100;
                column.Formatting = new XlCellFormatting();
                column.Formatting.NumberFormat = @"_([$-409]* #,##0.00_);_([$-409]* \(#,##0.00\);_([$-409]*
            }
        }
        // Finalize the group creation.
        sheet.EndGroup();
        // Create the column "F", adjust its width and set the specific number format for its cells.
        using (IXlColumn column = sheet.CreateColumn()) {
            column.WidthInPixels = 100;
            column.Formatting = new XlCellFormatting();
            column.Formatting.NumberFormat = @"_([$-409]* #,##0.00_);_([$-409]* \(#,##0.00\);_([$-409]*
        }
        // Specify formatting settings for cells containing data.
        XlCellFormatting rowFormatting = new XlCellFormatting();
        rowFormatting.Font = XlFont.BodyFont();
        rowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Light1, 0.0));
        // Specify formatting settings for the header rows.
        XlCellFormatting headerRowFormatting = new XlCellFormatting();
        headerRowFormatting.Font = XlFont.BodyFont();
        headerRowFormatting.Font.Bold = true;
        headerRowFormatting.Font.Color = XlColor.FromTheme(XlThemeColor.Light1, 0.0);
        headerRowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent1, 0.0));
        // Specify formatting settings for the total rows.
        XlCellFormatting totalRowFormatting = new XlCellFormatting();
        totalRowFormatting.Font = XlFont.BodyFont();
        totalRowFormatting.Font.Bold = true;
        totalRowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Light2, 0.0));
        // Specify formatting settings for the grand total row.
        XlCellFormatting grandTotalRowFormatting = new XlCellFormatting();
        grandTotalRowFormatting.Font = XlFont.BodyFont();
        grandTotalRowFormatting.Font.Bold = true;
        grandTotalRowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Light2, -0.2));
        // Generate data for the document.
        Random random = new Random();
        string[] products = new string[] { "Camembert Pierrot", "Gorgonzola Telino", "Mascarpone Fabioli",
        // Begin to group worksheet rows (create the outer group of rows).
        sheet.BeginGroup(false);
        for (int p = 0; p < 2; p++) {
            // Generate the header row.
            using (IXlRow row = sheet.CreateRow()) {
                using (IXlCell cell = row.CreateCell()) {
                    cell.Value = (p == 0) ? "East" : "West";
                    cell.ApplyFormatting(headerRowFormatting);
                    cell.Formatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent2, 0.0));
                }
            }
        }
    }
}

```

```

    }
    for(int i = 0; i < 4; i++) {
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = string.Format("Q{0}", i + 1);
            cell.ApplyFormatting(headerRowFormatting);
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Right, XlVerticalAl
        }
    }
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Yearly total";
        cell.ApplyFormatting(headerRowFormatting);
        cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizontalAlignment.Right, XlVerticalAl
    }
}
// Create and group data rows (create the inner group of rows containing sales data for the sp
sheet.BeginGroup(false);
for(int i = 0; i < 4; i++) {
    using (IXlRow row = sheet.CreateRow()) {
        using (IXlCell cell = row.CreateCell()) {
            cell.Value = products[i];
            cell.ApplyFormatting(rowFormatting);
            cell.Formatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent2, 0
        }
        for(int j = 0; j < 4; j++) {
            using (IXlCell cell = row.CreateCell()) {
                cell.Value = Math.Round(random.NextDouble() * 2000 + 3000);
                cell.ApplyFormatting(rowFormatting);
            }
        }
        using (IXlCell cell = row.CreateCell()) {
            cell.SetFormula(XlFunc.Sum(XlCellRange.FromLTRB(1, row.RowIndex, 4, row.RowIndex));
            cell.ApplyFormatting(rowFormatting);
            cell.ApplyFormatting(XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Light2, 0.0));
        }
    }
}
// Finalize the group creation.
sheet.EndGroup();
// Create the total row.
using (IXlRow row = sheet.CreateRow()) {
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Total";
        cell.ApplyFormatting(totalRowFormatting);
        cell.Formatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent2, 0.6));
    }
    for(int j = 0; j < 5; j++) {
        using (IXlCell cell = row.CreateCell()) {
            cell.SetFormula(XlFunc.Subtotal(XlCellRange.FromLTRB(j + 1, row.RowIndex - 4, j +
            cell.ApplyFormatting(totalRowFormatting);
        }
    }
}
// Finalize the group creation.
sheet.EndGroup();
// Create the grand total row.
using (IXlRow row = sheet.CreateRow()) {
    using (IXlCell cell = row.CreateCell()) {
        cell.Value = "Grand total";
        cell.ApplyFormatting(grandTotalRowFormatting);
        cell.Formatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeColor.Accent2, 0.4));
    }
    for(int j = 0; j < 5; j++) {
        using (IXlCell cell = row.CreateCell()) {
            cell.SetFormula(XlFunc.Subtotal(XlCellRange.FromLTRB(j + 1, 1, j + 1, row.RowIndex -

```

```
        cell.ApplyFormatting(grandTotalRowFormatting);  
    }  
    }  
    }  
}
```

Visual Basic

```

(DataActions.vb)
' Create an exporter instance.
Dim exporter As IXlExporter = XlExport.CreateExporter(documentFormat)
' Create a new document.
Using document As IXlDocument = exporter.CreateDocument(stream)
    document.Options.Culture = CultureInfo.CurrentCulture
    ' Create a worksheet.
    Using sheet As IXlSheet = document.CreateSheet()
        ' Specify the summary row and summary column location for the group
        sheet.OutlineProperties.SummaryBelow = True
        sheet.OutlineProperties.SummaryRight = True
        ' Create the column "A" and set its width.
        Using column As IXlColumn = sheet.CreateColumn()
            column.WidthInPixels = 200
        End Using
        ' Begin to group worksheet columns starting from the column "B" to
        sheet.BeginGroup(False)
        ' Create four successive columns ("B", "C", "D" and "E") and set t
        For i As Integer = 0 To 3
            Using column As IXlColumn = sheet.CreateColumn()
                column.WidthInPixels = 100
                column.Formatting = New XlCellFormatting()
                column.Formatting.NumberFormat = "_([$$-409]* #,##0.00_);_
            End Using
        Next i
        ' Finalize the group creation.
        sheet.EndGroup()
        ' Create the column "F", adjust its width and set the specific num
        Using column As IXlColumn = sheet.CreateColumn()
            column.WidthInPixels = 100
            column.Formatting = New XlCellFormatting()
            column.Formatting.NumberFormat = "_([$$-409]* #,##0.00_);_([$$
        End Using
        ' Specify formatting settings for cells containing data.
        Dim rowFormatting As New XlCellFormatting()
        rowFormatting.Font = XlFont.BodyFont()
        rowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThemeCol
        ' Specify formatting settings for the header rows.
        Dim headerRowFormatting As New XlCellFormatting()
        headerRowFormatting.Font = XlFont.BodyFont()
        headerRowFormatting.Font.Bold = True
        headerRowFormatting.Font.Color = XlColor.FromTheme(XlThemeColor.Li
        headerRowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlTh
        ' Specify formatting settings for the total rows.
        Dim totalRowFormatting As New XlCellFormatting()
        totalRowFormatting.Font = XlFont.BodyFont()
        totalRowFormatting.Font.Bold = True
        totalRowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(XlThe
        ' Specify formatting settings for the grand total row.
        Dim grandTotalRowFormatting As New XlCellFormatting()
        grandTotalRowFormatting.Font = XlFont.BodyFont()

```

```

grandTotalRowFormatting.Font.Bold = True
grandTotalRowFormatting.Fill = XlFill.SolidFill(XlColor.FromTheme(
' Generate data for the document.
Dim random As New Random()
Dim products() As String = { "Camembert Pierrot", "Gorgonzola Teli
' Begin to group worksheet rows (create the outer group of rows).
sheet.BeginGroup(False)
For p As Integer = 0 To 1
    ' Generate the header row.
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = If(p = 0, "East", "West")
            cell.ApplyFormatting(headerRowFormatting)
            cell.Formatting.Fill = XlFill.SolidFill(XlColor.FromTh
        End Using
        For i As Integer = 0 To 3
            Using cell As IXlCell = row.CreateCell()
                cell.Value = String.Format("Q{0}", i + 1)
                cell.ApplyFormatting(headerRowFormatting)
                cell.ApplyFormatting(XlCellAlignment.FromHV(XlHori
            End Using
        Next i
        Using cell As IXlCell = row.CreateCell()
            cell.Value = "Yearly total"
            cell.ApplyFormatting(headerRowFormatting)
            cell.ApplyFormatting(XlCellAlignment.FromHV(XlHorizont
        End Using
    End Using
    ' Create and group data rows (create the inner group of rows c
    sheet.BeginGroup(False)
    For i As Integer = 0 To 3
        Using row As IXlRow = sheet.CreateRow()
            Using cell As IXlCell = row.CreateCell()
                cell.Value = products(i)
                cell.ApplyFormatting(rowFormatting)
                cell.Formatting.Fill = XlFill.SolidFill(XlColor.Fr
            End Using
            For j As Integer = 0 To 3
                Using cell As IXlCell = row.CreateCell()
                    cell.Value = Math.Round(random.NextDouble() *
                    cell.ApplyFormatting(rowFormatting)
                End Using
            Next j
            Using cell As IXlCell = row.CreateCell()
                cell.SetFormula(XlFunc.Sum(XlCellRange.FromLTRB(1,
                cell.ApplyFormatting(rowFormatting)
                cell.ApplyFormatting(XlFill.SolidFill(XlColor.From
            End Using
        End Using
    Next i
    ' Finalize the group creation.

```

```

sheet.EndGroup()
' Create the total row.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Total"
        cell.ApplyFormatting(totalRowFormatting)
        cell.Formatting.Fill = XlFill.SolidFill(XlColor.FromTh
    End Using
    For j As Integer = 0 To 4
        Using cell As IXlCell = row.CreateCell()
            cell.SetFormula(XlFunc.Subtotal(XlCellRange.FromLT
            cell.ApplyFormatting(totalRowFormatting)
        End Using
    Next j
End Using
Next p
' Finalize the group creation.
sheet.EndGroup()
' Create the grand total row.
Using row As IXlRow = sheet.CreateRow()
    Using cell As IXlCell = row.CreateCell()
        cell.Value = "Grand total"
        cell.ApplyFormatting(grandTotalRowFormatting)
        cell.Formatting.Fill = XlFill.SolidFill(XlColor.FromTheme
    End Using
    For j As Integer = 0 To 4
        Using cell As IXlCell = row.CreateCell()
            cell.SetFormula(XlFunc.Subtotal(XlCellRange.FromLTRB(j
            cell.ApplyFormatting(grandTotalRowFormatting)
        End Using
    Next j
End Using
End Using
End Using

```

The following image illustrates the grouped worksheet data.

	A	B	C	D	E	F
1	East	Q1	Q2	Q3	Q4	Yearly total
6	Total	\$ 14,622.00	\$ 13,973.00	\$ 17,757.00	\$ 14,857.00	\$ 61,209.00
7	West	Q1	Q2	Q3	Q4	Yearly total
8	Camembert Pierrot	\$ 4,761.00	\$ 4,476.00	\$ 4,014.00	\$ 3,775.00	\$ 17,026.00
9	Gorgonzola Telino	\$ 3,625.00	\$ 4,956.00	\$ 4,417.00	\$ 4,808.00	\$ 17,806.00
10	Mascarpone Fabioli	\$ 3,726.00	\$ 3,633.00	\$ 3,213.00	\$ 4,974.00	\$ 15,546.00
11	Mozzarella di Giovanni	\$ 3,304.00	\$ 4,299.00	\$ 4,371.00	\$ 3,511.00	\$ 15,485.00
12	Total	\$ 15,416.00	\$ 17,364.00	\$ 16,015.00	\$ 17,068.00	\$ 65,863.00
13	Grand total	\$ 30,038.00	\$ 31,337.00	\$ 33,772.00	\$ 31,925.00	\$ 127,072.00

Filtering

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Filtering](#)

The [Excel Export Library](#) allows you to use the filtering functionality to filter large amounts of data in a worksheet and display only rows that meet the filtering criteria.

• **Filter a Cell Range**

Use the `IXISheet.AutoFilterColumns` property to get access to the worksheet's filter collection. Create an `XIFilterColumn` instance and add it to the collection to apply filtering to a specific column in a cell range. `XIFilterColumn.ColumnId` specifies the column's zero-based index in the filtered cell range. The `XIFilterColumn.FilterCriteria` property defines the column's filter criteria.

Call the `IXISheet.BeginFiltering` method to start filtering data using the specified filter expression(s). The method's `autoFilterRange` parameter specifies the header row of the cell range you wish to filter. This range automatically extends when you generate new rows. If a row contains cells that do not meet the filter criteria, the row is hidden. Call the `IXISheet.EndFiltering` method to finish filtering after generating all the data.

• **Filter a Table**

To filter data in a [table column](#), assign the required filter criteria to the column's `IXITableColumn.FilterCriteria` property before generating the table data.

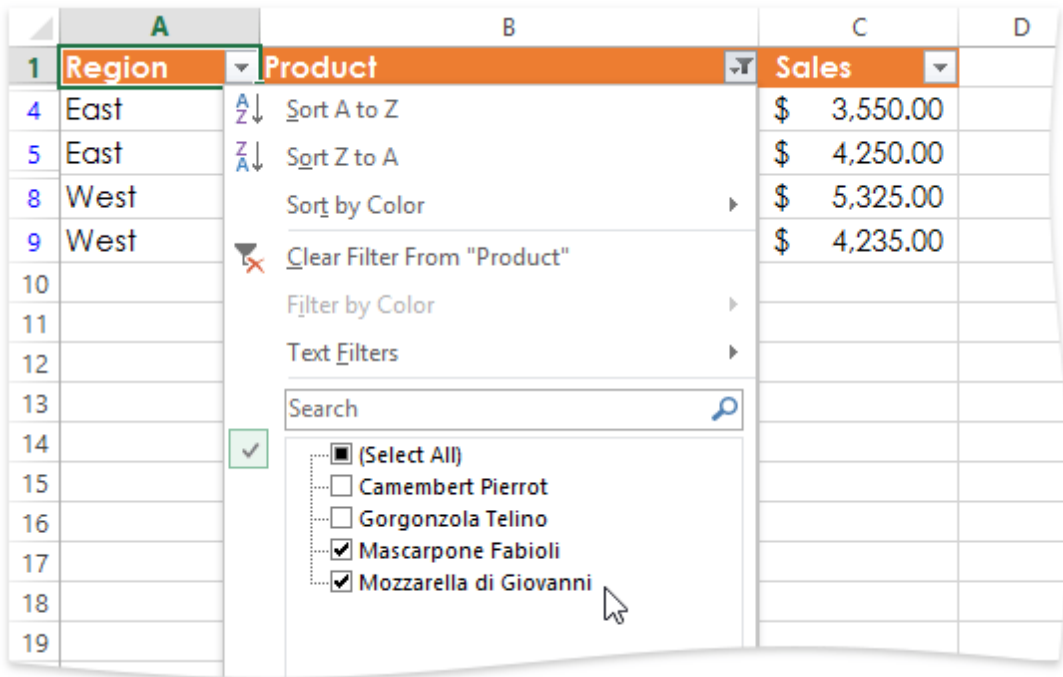
The following types of filters are available:

Filter Type	Description	Examples
XIValuesFilter	A filter by a list of cell values or date and time values.	Filter by Cell Values Filter by Date Values
XICustomFilters	A custom filter that uses filter values and comparison operators to construct the filter expression.	Apply a Custom Filter
XIDynamicFilter	A dynamic filter that shows dates that fall within a specified period, or displays values that are above or below average.	Apply a Dynamic Date Filter Filter Values that are Above or Below Average
XITop10Filter	A "Top 10" filter that displays the top/bottom ranked values.	Filter Top or Bottom Ranked Values
XIColorFilter	A filter by cell color or font color.	Filter by Cell Color

Use the `IXISheet.AutoFilterRange` property to enable filtering for a worksheet range without applying the filter criteria.

Filter by Cell Values

The example below demonstrates how to filter data in a column by a list of values.



Create an `XValuesFilter` object to specify the filter criteria. Use the **Add** method of the `XValuesFilter.Values` collection to specify cell values that should be included in the filtering results. You can include blank cells in a filter by setting the `XValuesFilter.FilterByBlank` property to **true**.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

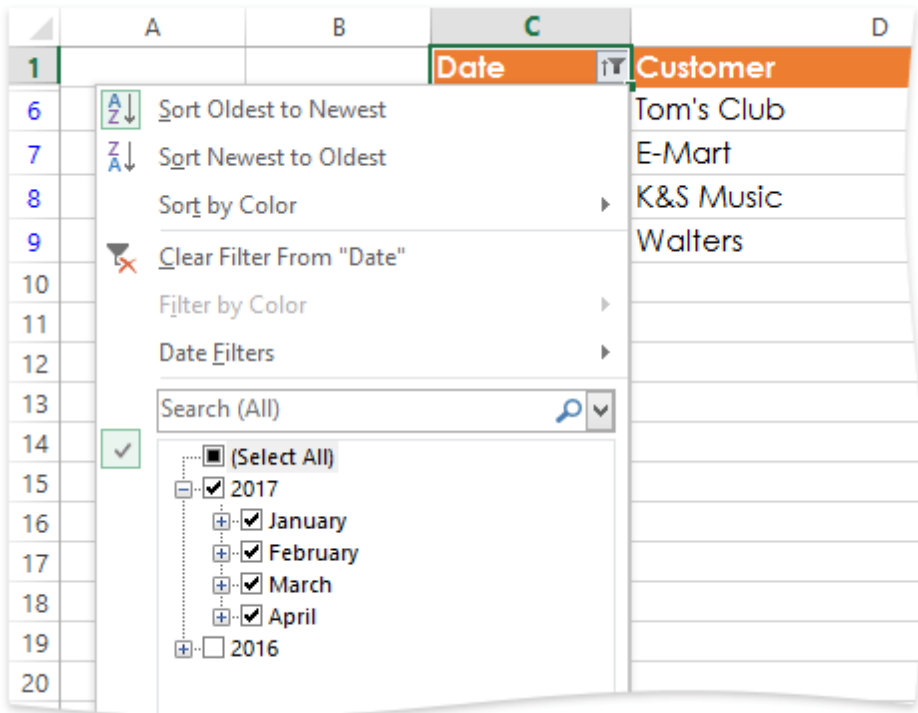
```
(DataActions.cs)
// Generate the header row.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new string[] { "Region", "Product", "Sales" }, headerRowFormatting);
// Start filtering data in the "Product" column by a list of values.
XlValuesFilter filter = new XlValuesFilter();
filter.Values.Add("Mascarpone Fabioli");
filter.Values.Add("Mozzarella di Giovanni");
sheet.AutoFilterColumns.Add(new XlFilterColumn(1, filter));
sheet.BeginFiltering(sheet.DataRange);
// Generate data for the document.
string[] products = new string[] { "Camembert Pierrot", "Gorgonzola Telino", "Mascarpone Fabioli", "Mozzaa
int[] amount = new int[] { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235 };
for (int i = 0; i < 8; i++)
{
    using (IXlRow row = sheet.CreateRow())
    {
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = (i < 4) ? "East" : "West";
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = products[i % 4];
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = amount[i];
            cell.ApplyFormatting(rowFormatting);
        }
    }
}
// Finish filtering.
sheet.EndFiltering();
```

Visual Basic

```
(DataActions.vb)
' Generate the header row.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New String() {"Region", "Product", "Sales"}, headerRowFormatting)
End Using
' Start filtering data in the "Product" column by a list of values.
Dim filter As New XlValuesFilter()
filter.Values.Add("Mascarpone Fabioli")
filter.Values.Add("Mozzarella di Giovanni")
sheet.AutoFilterColumns.Add(New XlFilterColumn(1, filter))
sheet.BeginFiltering(sheet.DataRange)
' Generate data for the document.
Dim products() As String = {"Camembert Pierrot", "Gorgonzola Telino", "Mascarpone Fabioli", "Mozzarella di"}
Dim amount() As Integer = {6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235}
For i As Integer = 0 To 7
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = If(i < 4, "East", "West")
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = products(i Mod 4)
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = amount(i)
            cell.ApplyFormatting(rowFormatting)
        End Using
    End Using
Next i
' Finish filtering.
sheet.EndFiltering()
```

Filter by Date Values

This example demonstrates how to filter date and time values in a column.



Create an `XIValuesFilter` object to specify the filter criteria. Use the `XIValuesFilter.DateGroups` collection's **Add** method to specify date and time values which should be used in the filter criteria. An instance of the `XIDateGroupItem` class defines each date and time filter value. When you create a new `XIDateGroupItem` object, you should specify the base date or time value to filter by and which part of this value to be used in the filter criteria.

The following example uses the `XIDateGroupItem` instance with the `XIDateGroupItem.Value` set to the current date and time, and `XIDateGroupItem.GroupingType` set to `XIDateTimeGroupingType.Year` to display the current year's dates:

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#	
----	--

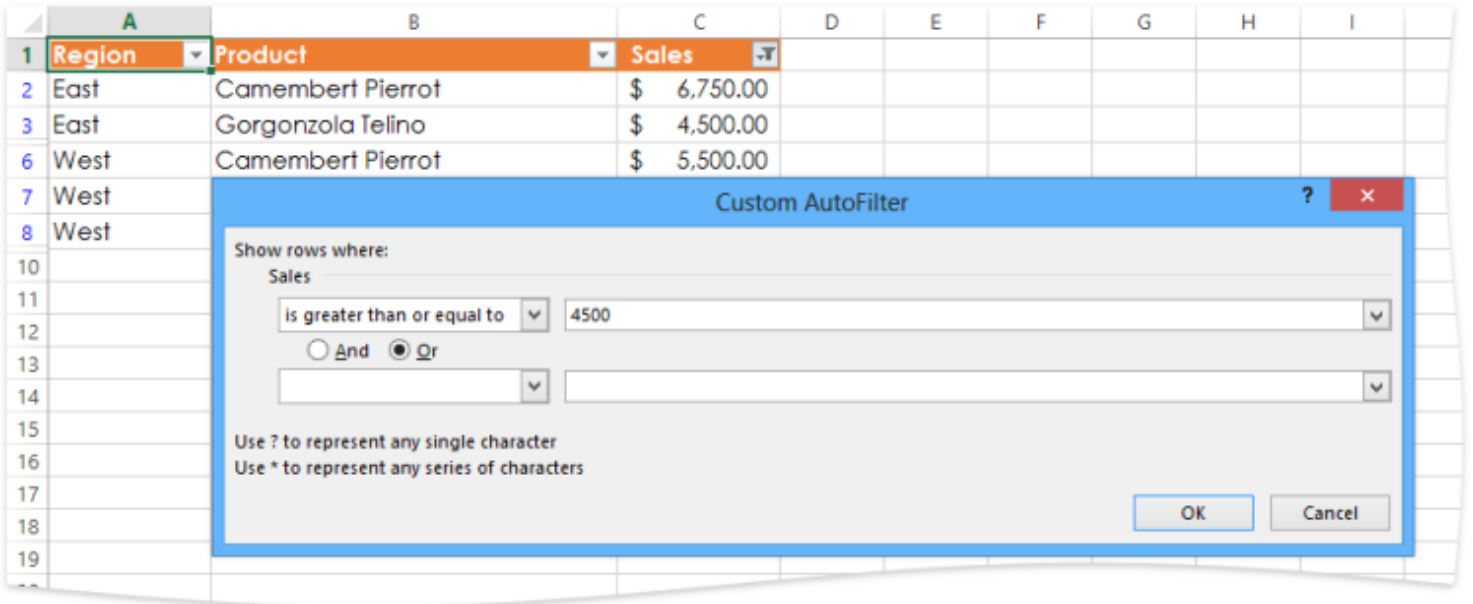
```
(DataActions.cs)
// Generate the header row.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new string[] { "Date", "Customer", "Total" }, headerRowF
// Create a date filter to display sales data for the current year.
XlValuesFilter filter = new XlValuesFilter();
filter.DateGroups.Add(new XlDateGroupItem(DateTime.Today, XlDateTimeGroupi
sheet.AutoFilterColumns.Add(new XlFilterColumn(0, filter));
sheet.BeginFiltering(sheet.DataRange);
// Generate data for the document.
string[] customers = new string[] { "Tom's Club", "E-Mart", "K&S Music", "
int[] amount = new int[] { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235
for (int i = 0; i < 8; i++)
{
    using (IXlRow row = sheet.CreateRow())
    {
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = (i < 4) ? new DateTime(DateTime.Today.AddYears(-1
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = customers[i % 4];
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = amount[i];
            cell.ApplyFormatting(rowFormatting);
        }
    }
}
// Finish filtering.
sheet.EndFiltering();
```

Visual Basic

```
(DataActions.vb)
' Generate the header row.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New String() {"Date", "Customer", "Total"}, headerRowFor
End Using
' Create a date filter to display sales data for the current year.
Dim filter As New XlValuesFilter()
filter.DateGroups.Add(New XlDateGroupItem(Date.Today, XlDateTimeGroupingTy
sheet.AutoFilterColumns.Add(New XlFilterColumn(0, filter))
sheet.BeginFiltering(sheet.DataRange)
' Generate data for the document.
Dim customers() As String = {"Tom's Club", "E-Mart", "K&S Music", "Walters
Dim amount() As Integer = {6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235}
For i As Integer = 0 To 7
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = If(i < 4, New Date(Date.Today.AddYears(-1).Year,
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = customers(i Mod 4)
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = amount(i)
            cell.ApplyFormatting(rowFormatting)
        End Using
    End Using
Next i
' Finish filtering.
sheet.EndFiltering()
```

Apply a Custom Filter

The example below demonstrates how to specify the custom filter criteria to filter numbers in a column.



Create an `XICustomFilters` object to construct the filter expression. An instance of the `XICustomFilterCriteria` class defines a filter criterion for a custom filter. It includes a filter value (`XICustomFilterCriteria.Value`) and a comparison operator (`XICustomFilterCriteria.FilterOperator`). If you create a complex filter expression containing two filter criteria, specify the logical operator ("AND" or "OR") used to combine these criteria (`XICustomFilters.And`).

Tip

You can use the *wildcard* characters when performing text filtering. The asterisk `*` matches any number of characters, while the question mark `?` represents a single character. If you need to filter values containing a specific character, such as the asterisk, question mark or tilde, put the tilde (`~`) before it.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#	
----	--

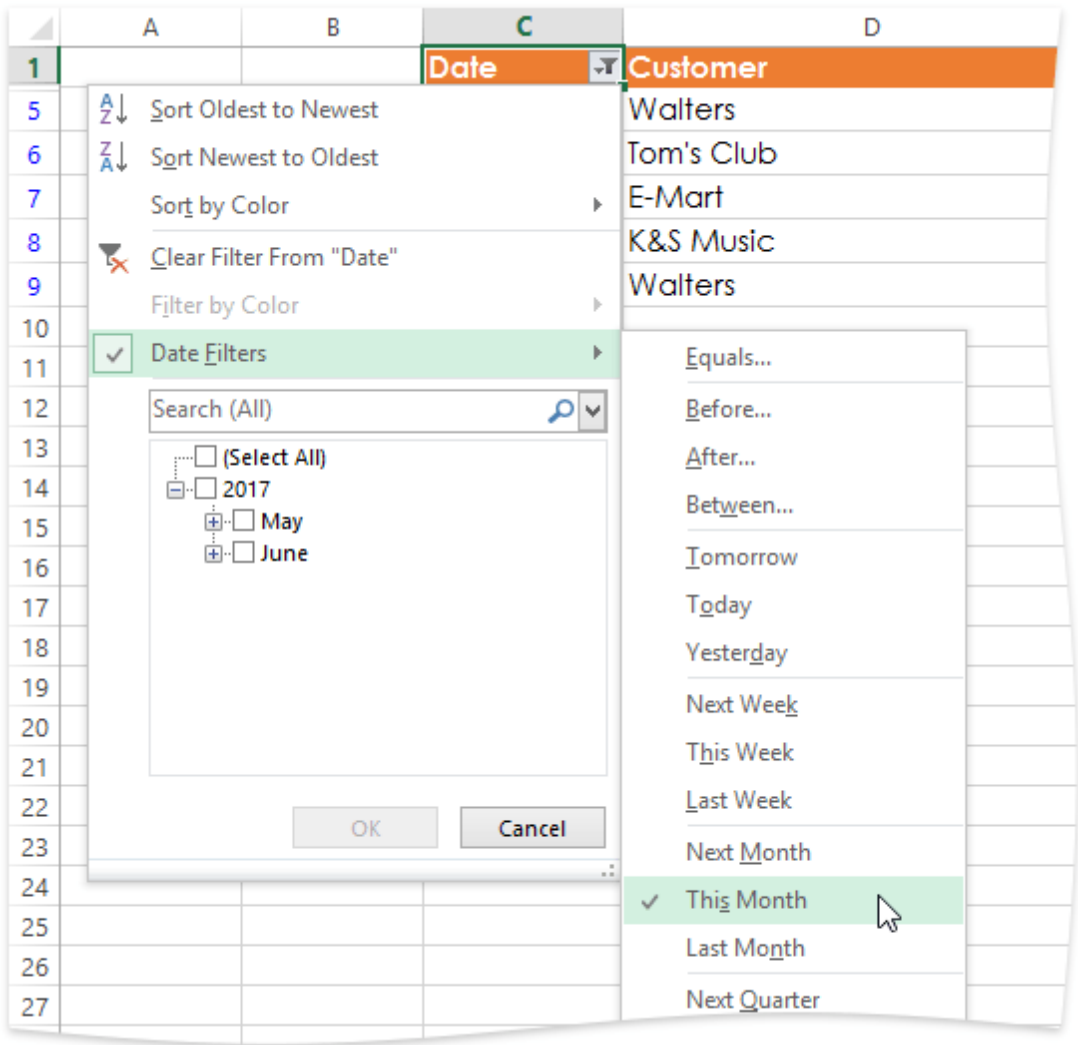
```
(DataActions.cs)
// Generate the header row.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new string[] { "Region", "Product", "Sales" }, headerRow)
// Create a custom filter to display values in the "Sales" column that are
XlCustomFilters filter = new XlCustomFilters(new XlCustomFilterCriteria(Xl
sheet.AutoFilterColumns.Add(new XlFilterColumn(2, filter));
// Start filtering data.
sheet.BeginFiltering(sheet.DataRange);
// Generate data for the document.
string[] products = new string[] { "Camembert Pierrot", "Gorgonzola Telino",
int[] amount = new int[] { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235
for (int i = 0; i < 8; i++)
{
    using (IXlRow row = sheet.CreateRow())
    {
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = (i < 4) ? "East" : "West";
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = products[i % 4];
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = amount[i];
            cell.ApplyFormatting(rowFormatting);
        }
    }
}
// Finish filtering.
sheet.EndFiltering();
```

Visual Basic

```
(DataActions.vb)
' Generate the header row.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New String() { "Region", "Product", "Sales" }, headerRow)
End Using
' Create a custom filter to display values in the "Sales" column that are
Dim filter As New XlCustomFilters(New XlCustomFilterCriteria(XlFilterOpera
sheet.AutoFilterColumns.Add(New XlFilterColumn(2, filter))
' Start filtering data.
sheet.BeginFiltering(sheet.DataRange)
' Generate data for the document.
Dim products() As String = { "Camembert Pierrot", "Gorgonzola Telino", "Ma
Dim amount() As Integer = { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235
For i As Integer = 0 To 7
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = If(i < 4, "East", "West")
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = products(i Mod 4)
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = amount(i)
            cell.ApplyFormatting(rowFormatting)
        End Using
    End Using
Next i
' Finish filtering.
sheet.EndFiltering()
```

Apply a Dynamic Date Filter

The following example demonstrates how to use a dynamic filter to show dates that occurred this month.



Create an `XIDynamicFilter` object to construct the filter criteria. Use the appropriate `XIDynamicFilterType` enumeration member to specify the dynamic filter type you wish to apply.

Note

The dynamic filter criteria can change when the data to which the filter is applied or the current system date changes.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

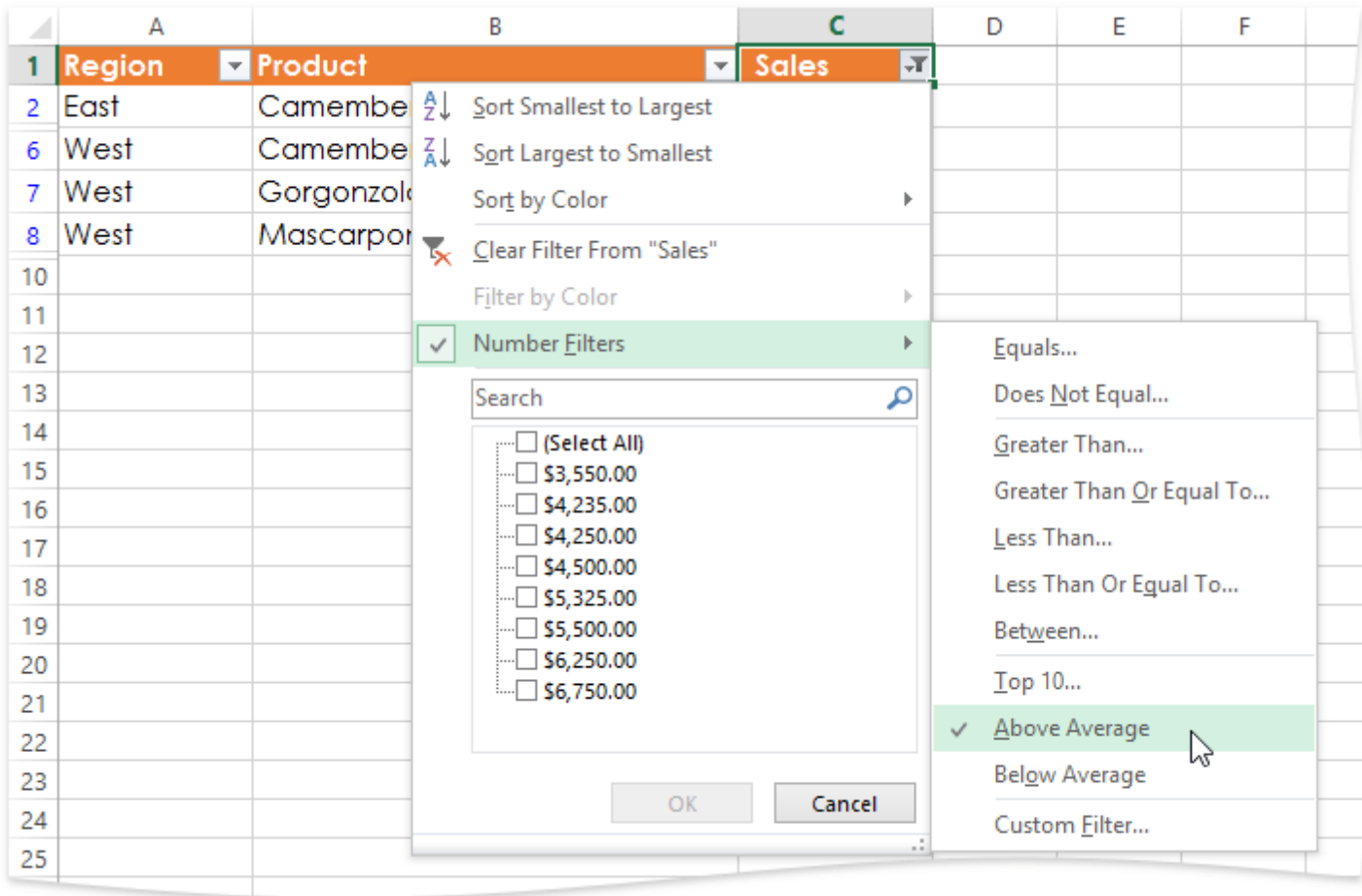
```
(DataActions.cs)
// Generate the header row.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new string[] { "Date", "Customer", "Total" }, headerRowF
// Create a dynamic filter to display dates that occur this month.
XlDynamicFilter filter = new XlDynamicFilter(XlDynamicFilterType.ThisMonth
sheet.AutoFilterColumns.Add(new XlFilterColumn(0, filter));
// Start filtering data.
sheet.BeginFiltering(sheet.DataRange);
// Generate data for the document.
string[] customers = new string[] { "Tom's Club", "E-Mart", "K&S Music", "
int[] amount = new int[] { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235
for (int i = 0; i < 8; i++)
{
    using (IXlRow row = sheet.CreateRow())
    {
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = DateTime.Now.AddDays(-7 * (7 - i));
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = customers[i % 4];
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = amount[i];
            cell.ApplyFormatting(rowFormatting);
        }
    }
}
// Finish filtering.
sheet.EndFiltering();
```

Visual Basic

```
(DataActions.vb)
' Generate the header row.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New String() {"Date", "Customer", "Total"}, headerRowFor
End Using
' Create a dynamic filter to display dates that occur this month.
Dim filter As New XlDynamicFilter(XlDynamicFilterType.ThisMonth)
sheet.AutoFilterColumns.Add(New XlFilterColumn(0, filter))
' Start filtering data.
sheet.BeginFiltering(sheet.DataRange)
' Generate data for the document.
Dim customers() As String = {"Tom's Club", "E-Mart", "K&S Music", "Walters
Dim amount() As Integer = {6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235}
For i As Integer = 0 To 7
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = Date.Now.AddDays(-7 * (7 - i))
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = customers(i Mod 4)
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = amount(i)
            cell.ApplyFormatting(rowFormatting)
        End Using
    End Using
Next i
' Finish filtering.
sheet.EndFiltering()
```

Filter Values that are Above or Below Average

The example below demonstrates how to use a dynamic filter to display sales values that are above average.



Create an `XIDynamicFilter` class instance to specify the filter criteria. Its constructor requires the following parameters:

- The dynamic filter type (`XIDynamicFilterType.AboveAverage` or `XIDynamicFilterType.BelowAverage`);
- The arithmetic mean of values in the filtered column that should be used to perform the comparison for the filter, since an exporter cannot calculate this value automatically.

C#

```
// Generate the header row.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new string[] { "Region", "Product", "Sales" }, headerRow);
// Apply a dynamic filter to the "Sales" column to display only values that are above average.
XlDynamicFilter filter = new XlDynamicFilter(XlDynamicFilterType.AboveAverage);
sheet.AutoFilterColumns.Add(new XlFilterColumn(2, filter));
sheet.BeginFiltering(sheet.DataRange);
// Generate data for the document.
string[] products = new string[] { "Camembert Pierrot", "Gorgonzola Telino", "Mascarpone", "Ricotta", "Taleggio", "Gorgonzola", "Mascarpone", "Ricotta" };
int[] amount = new int[] { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235 };
for (int i = 0; i < 8; i++)
{
    using (IXlRow row = sheet.CreateRow())
    {
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = (i < 4) ? "East" : "West";
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = products[i % 4];
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = amount[i];
            cell.ApplyFormatting(rowFormatting);
        }
    }
}
// Finish filtering.
sheet.EndFiltering();
```

Visual Basic

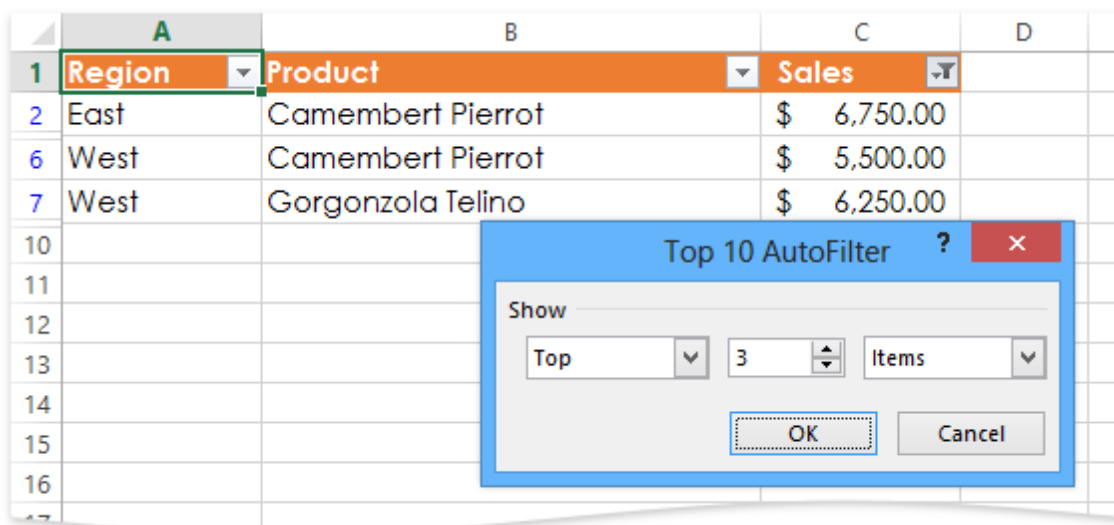
```

' Generate the header row.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New String() { "Region", "Product", "Sales" }, headerRow)
End Using
' Apply a dynamic filter to the "Sales" column to display only values that
Dim filter As New XlDynamicFilter(XlDynamicFilterType.AboveAverage, 5045)
sheet.AutoFilterColumns.Add(New XlFilterColumn(2, filter))
sheet.BeginFiltering(sheet.DataRange)
' Generate data for the document.
Dim products() As String = { "Camembert Pierrot", "Gorgonzola Telino", "Ma
Dim amount() As Integer = { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235
For i As Integer = 0 To 7
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = If(i < 4, "East", "West")
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = products(i Mod 4)
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = amount(i)
            cell.ApplyFormatting(rowFormatting)
        End Using
    End Using
Next i
' Finish filtering.
sheet.EndFiltering()

```

Filter Top or Bottom Ranked Values

This example demonstrates how to display three products with the highest sales using the "Top 10" filter.



Create an `XITop10Filter` instance to specify the filter criteria. This filter type requires the following parameters:

- A number or percentage of column items to display (`XITop10Filter.Value`);
- A filter value based on which to perform the comparison for the filter (`XITop10Filter.FilterValue`);
If you filter for the top (bottom) items, all column values that are greater than (less than) or equal to the specified filter value are shown.
- A value indicating whether to filter column values by top or bottom order (`XITop10Filter.Top`);
- A value indicating whether to show values that fall in the top/bottom N percent of the column (`XITop10Filter.Percent`).

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#	
----	--

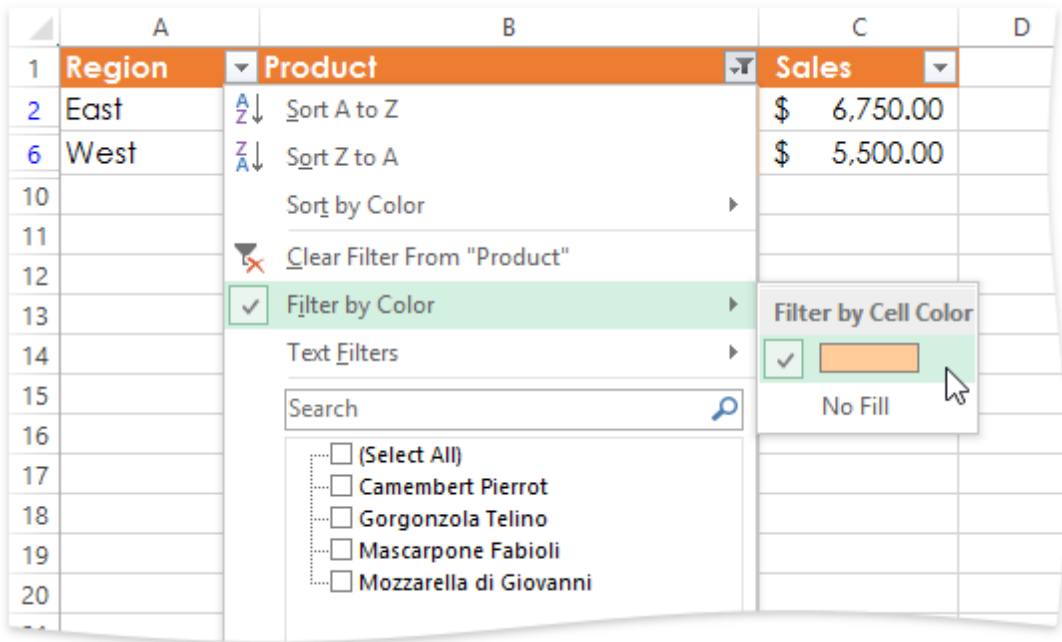
```
(DataActions.cs)
// Generate the header row.
using (IXlRow row = sheet.CreateRow())
    row.BulkCells(new string[] { "Region", "Product", "Sales" }, headerRow)
// Create a Top 10 filter to display three products with the highest sales
XlTop10Filter filter = new XlTop10Filter(3, 5500, true, false);
sheet.AutoFilterColumns.Add(new XlFilterColumn(2, filter));
// Start filtering data.
sheet.BeginFiltering(sheet.DataRange);
// Generate data for the document.
string[] products = new string[] { "Camembert Pierrot", "Gorgonzola Telino", "Mascarpone", "Ricotta", "Tiramisu", "Zabaglione", "Panna Cotta", "Cannoli" };
int[] amount = new int[] { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235 };
for (int i = 0; i < 8; i++)
{
    using (IXlRow row = sheet.CreateRow())
    {
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = (i < 4) ? "East" : "West";
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = products[i % 4];
            cell.ApplyFormatting(rowFormatting);
        }
        using (IXlCell cell = row.CreateCell())
        {
            cell.Value = amount[i];
            cell.ApplyFormatting(rowFormatting);
        }
    }
}
// Finish filtering.
sheet.EndFiltering();
```

Visual Basic

```
(DataActions.vb)
' Generate the header row.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New String() {"Region", "Product", "Sales"}, headerRowFo
End Using
' Create a Top 10 filter to display three products with the highest sales.
Dim filter As New XlTop10Filter(3, 5500, True, False)
sheet.AutoFilterColumns.Add(New XlFilterColumn(2, filter))
' Start filtering data.
sheet.BeginFiltering(sheet.DataRange)
' Generate data for the document.
Dim products() As String = {"Camembert Pierrot", "Gorgonzola Telino", "Mas
Dim amount() As Integer = {6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235}
For i As Integer = 0 To 7
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = If(i < 4, "East", "West")
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = products(i Mod 4)
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = amount(i)
            cell.ApplyFormatting(rowFormatting)
        End Using
    End Using
Next i
' Finish filtering.
sheet.EndFiltering()
```

Filter by Cell Color

The following example demonstrates how to filter cells in a column by a fill color.



Create an `XIColorFilter` class instance and use its properties to specify the filter criteria.

- `XIColorFilter.Color` - specifies the color to filter by;
- `XIColorFilter.FilterByCellColor` - specifies whether the filter should use the cell color or font color as a criterion.

You can also filter a column by a pattern fill applied to its cells by additionally specifying the `XIColorFilter.PatternType` and `XIColorFilter.PatternColor` properties of the pattern fill you wish to use as a criterion.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#


```
(DataActions.vb)
' Generate the header row.
Using row As IXlRow = sheet.CreateRow()
    row.BulkCells(New String() { "Region", "Product", "Sales" }, headerRow)
End Using
' Start filtering data in the "Product" column by the specified fill color
Dim filter As New XlColorFilter()
filter.Color = XlColor.FromArgb(&Hffcc99)
filter.FilterByCellColor = True
sheet.AutoFilterColumns.Add(New XlFilterColumn(1, filter))
sheet.BeginFiltering(sheet.DataRange)
' Generate data for the document.
Dim products() As String = { "Camembert Pierrot", "Gorgonzola Telino", "Ma
Dim amount() As Integer = { 6750, 4500, 3550, 4250, 5500, 6250, 5325, 4235
For i As Integer = 0 To 7
    Using row As IXlRow = sheet.CreateRow()
        Using cell As IXlCell = row.CreateCell()
            cell.Value = If(i < 4, "East", "West")
            cell.ApplyFormatting(rowFormatting)
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = products(i Mod 4)
            cell.ApplyFormatting(rowFormatting)
            If i Mod 4 = 0 Then
                cell.ApplyFormatting(XlFill.SolidFill(XlColor.FromArgb(&Hf
            End If
        End Using
        Using cell As IXlCell = row.CreateCell()
            cell.Value = amount(i)
            cell.ApplyFormatting(rowFormatting)
        End Using
    End Using
Next i
' Finish filtering.
sheet.EndFiltering()
```

Sparklines

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Sparklines](#)

This section contains the following examples:

- [How to: Create Sparklines](#)
- [How to: Customize the Sparkline Appearance](#)
- [How to: Specify Sparkline Axis Settings](#)

How to: Create Sparklines

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Sparklines](#) > [How to: Create Sparklines](#)

Worksheet sparklines are organized in groups. Each group contains one or more sparklines of the same type that share identical format settings and axis scaling options. To create a new group of sparklines, use the `XISparklineGroup` constructor which requires the following parameters:

- **Data Range.** A `XICellRange` object that specifies the data source for the sparkline group. Note that a data range for a single sparkline should be a continuous one-dimensional cell range (all cells of which are located in a single row or column). The number of the ranges must be equal to the number of cells in the location range. If the specified cell range is insufficient for a sparkline group, an exception is thrown.
- **Location Range.** A `XICellRange` object that specifies a range of cells where the sparkline group should be located. The dimension of the location range must be the same as the corresponding dimension of the data range, otherwise an exception is thrown.

You do not have to create individual sparklines in a group, they are created automatically with this constructor. All created sparklines have the default `XISparklineType.Line` type.

To add a sparkline group to the worksheet, use the **Add** method of the collection available with the `XISparklineGroup.Sparklines` property, as illustrated in the code snippet below:

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(SparklineActions.cs)
// Create a group of line sparklines.
XISparklineGroup group = new XISparklineGroup(XICellRange.FromLTRB(1, 1, 4, 6), XICellRange.FromLTRB(5, 1, 6, 6))
// Set the sparkline weight.
group.LineWeight = 1.25;
// Display data markers on the sparklines.
group.DisplayMarkers = true;
sheet.SparklineGroups.Add(group);
```

Visual Basic

```
(SparklineActions.vb)
' Create a group of line sparklines.
Dim group As New XISparklineGroup(XICellRange.FromLTRB(1, 1, 4, 6), XICellRange.FromLTRB(5, 1, 6, 6))
' Set the sparkline weight.
group.LineWeight = 1.25
' Display data markers on the sparklines.
group.DisplayMarkers = True
sheet.SparklineGroups.Add(group)
```

Note

If you create a sparkline group using the default parameterless constructor, the group does not contain sparklines. You should manually create a `XISparkline` object and add it to the collection accessible with the `XISparklineGroup.Sparklines` property.

How to: Customize the Sparkline Appearance

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Sparklines](#) > [How to: Customize the Sparkline Appearance](#)

Specify a Series Color

To specify a color used to draw a series in each sparkline in the sparkline group, use the `XISparklineGroup.ColorSeries` property. If sparklines are of the **Line** type (the `XISparklineGroup.SparklineType` property is equal to the `XISparklineType.Line` value) you can also set the line weight using the `XISparklineGroup.LineWeight` property.


Highlight Data Points

You can highlight important data points on a sparkline by displaying and coloring markers on *line* sparklines, or by differentiating points by color on *column* and *win/loss* sparklines.

The table below lists the sparkline group properties used to highlight specific points on each sparkline in the sparkline group.

Property	Description
<code>XISparklineGroup.ColorMarker</code>	Gets or sets the color of the data markers for each sparkline in the sparkline group.
<code>XISparklineGroup.DisplayMarkers</code>	Gets or sets whether data markers are displayed for each sparkline in the sparkline group.
<code>XISparklineGroup.HighlightFirst</code>	Gets or sets whether the first data point in the sparkline should be colored differently.
<code>XISparklineGroup.ColorLast</code>	Gets or sets the color of the last data point for each sparkline in the sparkline group.
<code>XISparklineGroup.HighlightLast</code>	Gets or sets whether the last data point in the sparkline should be colored differently.
<code>XISparklineGroup.ColorHigh</code>	Gets or sets the color of the highest data point for each sparkline in the sparkline group.
<code>XISparklineGroup.HighlightHighest</code>	Gets or sets whether the highest data point in the sparkline should be colored differently.
<code>XISparklineGroup.ColorLow</code>	Gets or sets the color of the lowest data point for each sparkline in the sparkline group.
<code>XISparklineGroup.HighlightLowest</code>	Gets or sets whether the lowest data point in the sparkline should be colored differently.
<code>XISparklineGroup.ColorNegative</code>	Gets or sets the color of the negative data points for each sparkline in the sparkline group.
<code>XISparklineGroup.HighlightNegative</code>	Gets or sets whether negative data points in the sparkline should be colored differently.

Example

 Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T253492 .	
C#	

```
(SparklineActions.cs)
// Create a sparkline group.
XlSparklineGroup group = new XlSparklineGroup(XlCellRange.FromLTRB(1, 1, 8, 6), XlCellRange.FromLTRB(9, 1,
// Change the sparkline group type to "Column".
group.SparklineType = XlSparklineType.Column;
// Set the series color.
group.ColorSeries = XlColor.FromTheme(XlThemeColor.Accent1, 0.0);
// Set the color for negative points on sparklines.
group.ColorNegative = XlColor.FromTheme(XlThemeColor.Accent2, 0.0);
// Set the color for the highest points on sparklines.
group.ColorHigh = XlColor.FromTheme(XlThemeColor.Accent6, 0.0);
// Highlight the highest and negative points on each sparkline in the group.
group.HighlightNegative = true;
group.HighlightHighest = true;
sheet.SparklineGroups.Add(group);
```

Visual Basic

```
(SparklineActions.vb)
' Create a sparkline group.
Dim group As New XlSparklineGroup(XlCellRange.FromLTRB(1, 1, 8, 6), XlCellRange.FromLTRB(9, 1, 9, 6))
' Change the sparkline group type to "Column".
group.SparklineType = XlSparklineType.Column
' Set the series color.
group.ColorSeries = XlColor.FromTheme(XlThemeColor.Accent1, 0.0)
' Set the color for negative points on sparklines.
group.ColorNegative = XlColor.FromTheme(XlThemeColor.Accent2, 0.0)
' Set the color for the highest points on sparklines.
group.ColorHigh = XlColor.FromTheme(XlThemeColor.Accent6, 0.0)
' Highlight the highest and negative points on each sparkline in the group.
group.HighlightNegative = True
group.HighlightHighest = True
sheet.SparklineGroups.Add(group)
```

How to: Specify Sparkline Axis Settings

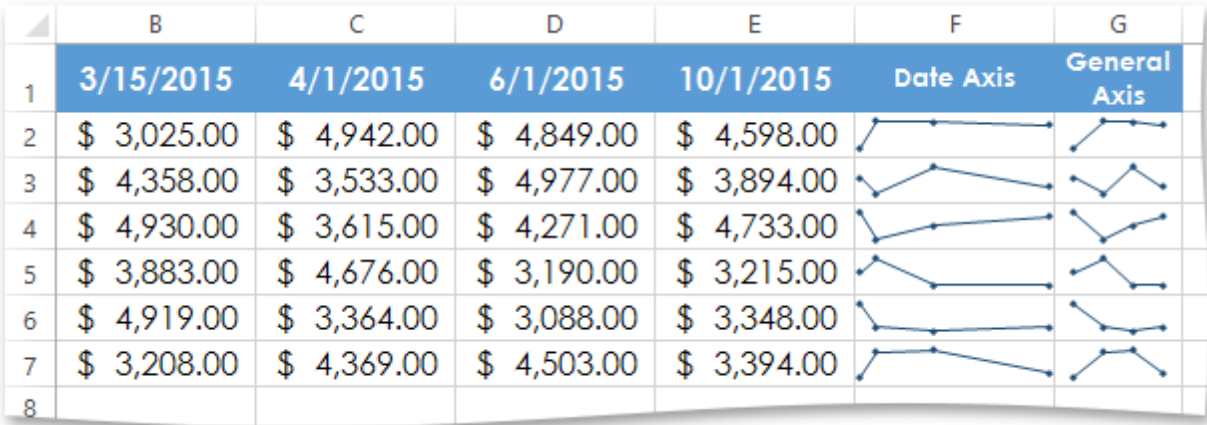
[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Sparklines](#) > [How to: Specify Sparkline Axis Settings](#)

Horizontal Axis Options

- Visibility**
Set the `XISparklineGroup.DisplayXAxis` property to **true** to display the horizontal axis on a sparkline. Use the `XISparklineGroup.ColorAxis` property to specify the axis color.
- Axis Type**
A sparkline uses the **general axis type** by default - the axis is inferred from the data range, and data points are displayed at regular intervals. However, if the underlying data occurs at irregular periods, you can reflect these time intervals on a sparkline using the **date axis type** that changes the space between data markers proportionally to the time intervals. Assign the cell range containing date values for the required sparkline group to the `XISparklineGroup.DateRange` property to change the axis type to date axis.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.



C#

```
(SparklineActions.cs)
// Create a group of line sparklines.
XISparklineGroup group = new XISparklineGroup(XlCellRange.Parse("B2:E7"), XlCellRange.Parse("F2:F7"));
// Specify the date range for the sparkline group.
group.DateRange = XlCellRange.Parse("B1:E1");
// Set the sparkline weight.
group.LineWeight = 1.25;
// Display data markers on the sparklines.
group.DisplayMarkers = true;
sheet.SparklineGroups.Add(group);
```

Visual Basic

```
(SparklineActions.vb)
' Create a group of line sparklines.
Dim group As New XISparklineGroup(XlCellRange.Parse("B2:E7"), XlCellRange.Parse("F2:F7"))
' Specify the date range for the sparkline group.
group.DateRange = XlCellRange.Parse("B1:E1")
' Set the sparkline weight.
group.LineWeight = 1.25
' Display data markers on the sparklines.
group.DisplayMarkers = True
sheet.SparklineGroups.Add(group)
```

Plotting Order

You can change the direction in which data is plotted in each sparkline of a sparkline group. Set the `XISparklineGroup.RightToLeft` property to **true** to display data in reverse (right-to-left) direction.

Vertical Axis Options

You can specify how to calculate the minimum and maximum values for the vertical axis using one of the following options:

- **Automatic.** Values are calculated individually for each sparkline in the group based on the lowest and highest values in the sparkline data range. Set the `XISparklineGroup.MinScaling` and `XISparklineGroup.MaxScaling` properties to the `XISparklineAxisScaling.Individual` value to use this option.
- **Identical.** All sparklines in the group use the same scale that is automatically calculated based on the lowest and highest values in the group data range. Set the `XISparklineGroup.MinScaling` and `XISparklineGroup.MaxScaling` properties to the `XISparklineAxisScaling.Group` value to use this option.
- **Custom.** Specifies custom values for the vertical axis. Set the `XISparklineGroup.MinScaling` and `XISparklineGroup.MaxScaling` properties to the `XISparklineAxisScaling.Custom` value to use custom scaling. Afterwards, assign the minimum value to the `XISparklineGroup.ManualMin` property and the maximum value to the `XISparklineGroup.ManualMax` property.

The example below demonstrates how to specify custom scaling values for a sparkline group's vertical axis.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#	
<pre>(SparklineActions.cs) // Create a sparkline group. XISparklineGroup group = new XISparklineGroup(XlCellRange.FromLTRB(1, 1, 4 // Set the sparkline color. group.ColorSeries = XlColor.FromTheme(XlThemeColor.Accent1, 0.0); // Change the sparkline group type to "Column". group.SparklineType = XISparklineType.Column; // Set the custom minimum value for the vertical axis. group.MinScaling = XISparklineAxisScaling.Custom; group.ManualMin = 1000.0; // Set the automatic maximum value for all sparklines in the group. group.MaxScaling = XISparklineAxisScaling.Group; sheet.SparklineGroups.Add(group);</pre>	
Visual Basic	

```
(SparklineActions.vb)
' Create a sparkline group.
Dim group As New XlSparklineGroup(XlCellRange.FromLTRB(1, 1, 4, 6), XlCell
' Set the sparkline color.
group.ColorSeries = XlColor.FromTheme(XlThemeColor.Accent1, 0.0)
' Change the sparkline group type to "Column".
group.SparklineType = XlSparklineType.Column
' Set the custom minimum value for the vertical axis.
group.MinScaling = XlSparklineAxisScaling.Custom
group.ManualMin = 1000.0
' Set the automatic maximum value for all sparklines in the group.
group.MaxScaling = XlSparklineAxisScaling.Group
sheet.SparklineGroups.Add(group)
```

Pictures

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Pictures](#)

This section contains the following examples:

- [How to: Insert and Position a Picture in a Worksheet](#)
- [How to: Add a Hyperlink to a Picture](#)

How to: Insert and Position a Picture in a Worksheet

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Pictures](#) > [How to: Insert and Position a Picture in a Worksheet](#)

The **Excel Export** library provides you with the capability to add images to a generated document. A picture is represented by the **IXIPicture** object that provides a set of properties and methods allowing you to insert the required image, adjust the picture placement and [add a hyperlink to a picture](#), if required.

To add a picture to a worksheet, do the following.

1. Create a new **IXIPicture** object by calling the **IXISheet.CreatePicture** method.
2. Set the **IXIPicture.Image** property to the [Image](#) object that specifies the image to be displayed in the worksheet. Use the **FromFile** or **FromStream** method of the **Image** object to load the required image from a file or data stream.
3. To specify the picture's position in the worksheet, use one of the following methods depending on the anchoring type you wish to use.

- **Absolute Anchoring**
Use the **IXIPicture.SetAbsoluteAnchor** method to specify the absolute position of the picture in the worksheet based on absolute coordinates that represent offsets from the left and top sides of the worksheet to the picture's top-left corner and set the desired width and height of the picture. The picture will not move or resize with the underlying cells.

- **One Cell Anchoring**
Use the **IXIPicture.SetOneCellAnchor** method to anchor the top-left corner of the picture to a cell in the worksheet and set the picture's width and height. The picture will move with the anchor cell, but its size will remain the same.

- **Two Cell Anchoring**
Use the **IXIPicture.SetTwoCellAnchor** method to anchor the picture to two specific cells in the worksheet. The first anchor cell defines the position of the top-left corner of the picture, while the second one specifies where picture's bottom-right corner is located. The last parameter of this method allows you to specify how the picture should behave when the underlying columns and rows are resized and moved.

Note that you can also use the **IXIPicture.AnchorType**, **IXIPicture.TopLeft** and **IXIPicture.BottomRight** properties to set or change the anchoring type and position of the picture's anchors in a worksheet. To specify or modify the anchoring behavior of a picture anchored to two cells in a worksheet, use the **IXIPicture.AnchorBehavior** property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

The example below demonstrates how to use the **IXIPicture.SetTwoCellAnchor** method to anchor a picture to two cells in a worksheet. The starting anchor for a picture is located at the intersection of the second column ("B") and the second row, with no offsets. The final anchor is located at the intersection of the seventh column ("G") and the twelfth row, with offsets from both the column and row. The **IXIPicture.AnchorBehavior** is set to **XlAnchorType.TwoCell**, so that the picture moves and resizes with its underlying cells.

C#

```
(PictureActions.cs)
// Create a worksheet.
using (IXlSheet sheet = document.CreateSheet())
{
    // Insert a picture from a file and anchor it to cells.
    using (IXlPicture picture = sheet.CreatePicture())
    {
        picture.Image = Image.FromFile(Path.Combine(imagesPath, "image1.jpg"));
        picture.SetTwoCellAnchor(new XlAnchorPoint(1, 1, 0, 0), new XlAnchorPoint(6, 11, 2, 15), XlAnchorType.TwoCell);
    }
}
```

Visual Basic

```
(PictureActions.vb)
' Create a worksheet.
Using sheet As IXlSheet = document.CreateSheet()
    ' Insert a picture from a file and anchor it to cells.
    Using picture As IXlPicture = sheet.CreatePicture()
        picture.Image = Image.FromFile(Path.Combine(imagesPath, "image1.jp
        picture.SetTwoCellAnchor(New XlAnchorPoint(1, 1, 0, 0), New XlAnch
    End Using
End Using
```

The following image shows the result of executing the code above (the workbook is opened in Microsoft® Excel®).



Fit a Picture Into a Cell

The IXlPicture object provides the IXlPicture.StretchToCell and IXlPicture.FitToCell methods that allow you to resize a picture to fit a specific cell in a worksheet.

- Use the IXlPicture.StretchToCell method to stretch a picture in a desired cell so that it fills the entire space of this cell. If the cell size changes, the picture size will be adjusted accordingly.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(PictureActions.cs)
// Insert a picture from a file and stretch it to fill the cell B2.
using (IXlPicture picture = sheet.CreatePicture())
{
    picture.Image = Image.FromFile(Path.Combine(imagesPath, "image1.jp
    picture.StretchToCell(new XlCellPosition(1, 1));
}
```

Visual Basic

```
(PictureActions.vb)
' Insert a picture from a file and stretch it to fill the cell B2.
Using picture As IXlPicture = sheet.CreatePicture()
    picture.Image = Image.FromFile(Path.Combine(imagesPath, "image1.jp
    picture.StretchToCell(New XlCellPosition(1, 1))
End Using
End Using
```

The following image shows the result of executing the code above (the workbook is opened in Microsoft® Excel®).



Use the IXlPicture.FitToCell method to rescale a picture to fit a cell of a specified size retaining the picture's proportions. The last parameter of this method allows you to specify whether the picture should be aligned at the center of the cell.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(PictureActions.cs)
// Insert a picture from a file to fit in the cell B2.
using (IXlPicture picture = sheet.CreatePicture())
{
    picture.Image = Image.FromFile(Path.Combine(imagesPath, "image1.jp
    picture.FitToCell(new XlCellPosition(1, 1), 300, 154, true);
}
}
```

Visual Basic

```
(PictureActions.vb)
' Insert a picture from a file to fit in the cell B2.
Using picture As IXlPicture = sheet.CreatePicture()
    picture.Image = Image.FromFile(Path.Combine(imagesPath, "image1.jp
    picture.FitToCell(New XlCellPosition(1, 1), 300, 154, True)
End Using
End Using
```

The following image shows the result of executing the code above (the workbook is opened in Microsoft® Excel®).



See Also

[How to: Add a Hyperlink to a Picture](#)

How to: Add a Hyperlink to a Picture

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Pictures](#) > [How to: Add a Hyperlink to a Picture](#)

This example demonstrates how to associate a hyperlink with a picture. Such hyperlinks are represented by the XIPictureHyperlink objects, which inherit basic hyperlink properties from the XIHyperlinkBase class.

To add a hyperlink to a picture, do the following.

- 1. [Create a picture](#) to which the hyperlink should be attached.
- 2. Access the XIPictureHyperlink object from the IXIPicture.HyperlinkClick property.
- 3. Use the **XIPictureHyperlink** object's XIHyperlinkBase.TargetUri property to specify the destination to which the hyperlink should refer.
- 4. Add a tooltip to your hyperlink, if required. To do this, assign the tooltip text to the XIHyperlinkBase.Tooltip property. This text will be displayed when the cursor hovers over the picture.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(PictureActions.cs)
// Create an exporter instance.
IXlExporter exporter = XlExport.CreateExporter(documentFormat);
// Create a new document.
using (IXlDocument document = exporter.CreateDocument(stream))
{
    document.Options.Culture = CultureInfo.CurrentCulture;
    // Create a worksheet.
    using (IXlSheet sheet = document.CreateSheet())
    {
        // Load a picture from a file and add a hyperlink to it.
        using (IXlPicture picture = sheet.CreatePicture())
        {
            picture.Image = Image.FromFile(Path.Combine(imagesPath, "DevExpress.png"));
            picture.HyperlinkClick.TargetUri = "http://www.devexpress.com";
            picture.HyperlinkClick.Tooltip = "Developer Express Inc.";
            picture.SetTwoCellAnchor(new XlAnchorPoint(1, 1, 0, 0), new XlAnchorPoint(10, 5, 2, 15), XlAnch
        }
    }
}
```

Visual Basic

```
(PictureActions.vb)
' Create an exporter instance.
Dim exporter As IXlExporter = XlExport.CreateExporter(documentFormat)
' Create a new document.
Using document As IXlDocument = exporter.CreateDocument(stream)
    document.Options.Culture = CultureInfo.CurrentCulture
    ' Create a worksheet.
    Using sheet As IXlSheet = document.CreateSheet()
        ' Load a picture from a file and add a hyperlink to it.
        Using picture As IXlPicture = sheet.CreatePicture()
            picture.Image = Image.FromFile(Path.Combine(imagesPath, "DevEx
            picture.HyperlinkClick.TargetUri = "http://www.devexpress.com"
            picture.HyperlinkClick.Tooltip = "Developer Express Inc."
            picture.SetTwoCellAnchor(New XlAnchorPoint(1, 1, 0, 0), New Xl
        End Using
    End Using
End Using
```

Printing

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Printing](#)

This section contains the following examples:

- [How to: Specify Print Settings](#)
- [How to: Set Page Margins](#)
- [How to: Add Headers and Footers to a Worksheet Printout](#)
- [How to: Insert Page Breaks in a Worksheet](#)
- [How to: Print Titles on a Worksheet](#)

How to: Specify Print Settings

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Printing](#) > [How to: Specify Print Settings](#)

The examples below demonstrate how to fine-tune print settings to create the required layout of a printed page.

- [Print Options](#)
- [Page Setup](#)

Print Options

To specify printing options for a worksheet, set the `IXISheet.PrintOptions` property to an instance of the `XIPrintOptions` class. Using the `XIPrintOptions` object's properties, you can print gridlines (`XIPrintOptions.GridLines`), add column headings to the top of the printout and row headings to the left side of the printout (`XIPrintOptions.Headings`), and center data on the printed page (`XIPrintOptions.HorizontalCentered` and `XIPrintOptions.VerticalCentered`).

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

The example below demonstrates how to specify print settings for a worksheet.

C#

```
(PageViewAndLayoutActions.cs)
// Specify print options for the worksheet.
sheet.PrintOptions = new XIPrintOptions();
// Print row and column headings.
sheet.PrintOptions.Headings = true;
// Print gridlines.
sheet.PrintOptions.GridLines = true;
// Center worksheet data on a printed page.
sheet.PrintOptions.HorizontalCentered = true;
sheet.PrintOptions.VerticalCentered = true;
```

Visual Basic

```
(PageViewAndLayoutActions.vb)
' Specify print options for the worksheet.
sheet.PrintOptions = New XIPrintOptions()
' Print row and column headings.
sheet.PrintOptions.Headings = True
' Print gridlines.
sheet.PrintOptions.GridLines = True
' Center worksheet data on a printed page.
sheet.PrintOptions.HorizontalCentered = True
sheet.PrintOptions.VerticalCentered = True
```

Page Setup

To define page formatting options, set the `IXISheet.PageSetup` property to an instance of the `XIPageSetup` class. The `XIPageSetup` object enables you to specify page orientation (`XIPageSetup.PageOrientation`), paper size (`XIPageSetup.PaperKind`), page order (`XIPageSetup.PagePrintOrder`), print quality (`XIPageSetup.HorizontalDpi` and `XIPageSetup.VerticalDpi`), number of copies (`XIPageSetup.Copies`) and other printing options.

To fit worksheet content into a specific number of pages, set the `XIPageSetup.FitToPage` property to **true** and specify the desired number of horizontal pages (`XIPageSetup.FitToWidth`) and vertical pages (`XIPageSetup.FitToHeight`) on which the worksheet should be printed. You can also scale the worksheet content arbitrarily by specifying the scale percentage using the `XIPageSetup.Scale` property. Note that this property takes effect only when the `XIPageSetup.FitToPage` property is **false**.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```
(PageViewAndLayoutActions.cs)
// Specify page settings for the worksheet.
sheet.PageSetup = new XlPageSetup();
// Select the paper size.
sheet.PageSetup.PaperKind = System.Drawing.Printing.PaperKind.A4;
// Set the page orientation to Landscape.
sheet.PageSetup.PageOrientation = XlPageOrientation.Landscape;
// Scale the print area to fit to one page wide.
sheet.PageSetup.FitToPage = true;
sheet.PageSetup.FitToWidth = 1;
sheet.PageSetup.FitToHeight = 0;
// Print in black and white.
sheet.PageSetup.BlackAndWhite = true;
// Specify the number of copies.
sheet.PageSetup.Copies = 2;
```

Visual Basic

```
(PageViewAndLayoutActions.vb)
' Specify page settings for the worksheet.
sheet.PageSetup = New XlPageSetup()
' Select the paper size.
sheet.PageSetup.PaperKind = System.Drawing.Printing.PaperKind.A4
' Set the page orientation to Landscape.
sheet.PageSetup.PageOrientation = XlPageOrientation.Landscape
' Scale the print area to fit to one page wide.
sheet.PageSetup.FitToPage = True
sheet.PageSetup.FitToWidth = 1
sheet.PageSetup.FitToHeight = 0
' Print in black and white.
sheet.PageSetup.BlackAndWhite = True
' Specify the number of copies.
sheet.PageSetup.Copies = 2
```

See Also

See the following examples to learn more about how to specify other printout options using the **Excel Export** API.

- [How to: Set Page Margins](#)
- [How to: Insert Page Breaks in a Worksheet](#)
- [How to: Print Titles on a Worksheet](#)
- [How to: Add Headers and Footers to a Worksheet Printout](#)

How to: Set Page Margins

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Printing](#) > [How to: Set Page Margins](#)

The example below demonstrates how to set margins for a printed page using properties of the `XlPageMargins` object. This object provides access to the margin settings and allows you to specify the size of the top (`XlPageMargins.Top`), bottom (`XlPageMargins.Bottom`), left (`XlPageMargins.Left`), right (`XlPageMargins.Right`), header (`XlPageMargins.Header`) and footer (`XlPageMargins.Footer`) margins. By default, the margin size is set in inches. To change the unit of margin measurement, use the `XlPageMargins.PageUnits` property.

To set margins for a particular worksheet, assign an instance of the `XlPageMargins` class to the `IXISheet.PageMargins` property.

Note

The `IXISheet.PageMargins` settings are used when worksheet pages are printed. To learn how to specify other print options for a worksheet, see the [How to: Specify Print Settings](#) document.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

(PageViewAndLayoutActions.cs)
sheet.PageMargins = new XlPageMargins();
// Set the unit of margin measurement.
sheet.PageMargins.PageUnits = XlPageUnits.Centimeters;
// Specify page margins.
sheet.PageMargins.Left = 2.0;
sheet.PageMargins.Right = 1.0;
sheet.PageMargins.Top = 1.25;
sheet.PageMargins.Bottom = 1.25;
// Specify header and footer margins.
sheet.PageMargins.Header = 0.7;
sheet.PageMargins.Footer = 0.7;

Visual Basic

(PageViewAndLayoutActions.vb)
sheet.PageMargins = New XlPageMargins()
' Set the unit of margin measurement.
sheet.PageMargins.PageUnits = XlPageUnits.Centimeters
' Specify page margins.
sheet.PageMargins.Left = 2.0
sheet.PageMargins.Right = 1.0
sheet.PageMargins.Top = 1.25
sheet.PageMargins.Bottom = 1.25
' Specify header and footer margins.
sheet.PageMargins.Header = 0.7
sheet.PageMargins.Footer = 0.7

How to: Add Headers and Footers to a Worksheet Printout

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Printing](#) > [How to: Add Headers and Footers to a Worksheet Printout](#)

The example below demonstrates how to insert headers and footers at the top and bottom of a worksheet printout.

To define the header and footer options, use the `IXISheet.HeaderFooter` property. It provides access to the `XIHeaderFooter` object, which allows you to define the header or footer for the first page (`XIHeaderFooter.FirstHeader` and `XIHeaderFooter.FirstFooter`), odd-numbered pages (`XIHeaderFooter.OddHeader` and `XIHeaderFooter.OddFooter`) and even-numbered pages (`XIHeaderFooter.EvenHeader` and `XIHeaderFooter.EvenFooter`) of the printed worksheet. Use the `XIHeaderFooter.FromLCR` method to specify the text that should be displayed in the left, center and right section of a header or footer, respectively.

Header and Footer Options

Use the properties of the `XIHeaderFooter` object to specify the following header/footer settings.

- `XIHeaderFooter.DifferentOddEven`

Set this property to **true** to specify different headers and footers for the odd-numbered and even-numbered pages. If this property is **false**, all pages in a worksheet have the same headers or footers as specified by the `XIHeaderFooter.OddHeader` and `XIHeaderFooter.OddFooter` properties.

- `XIHeaderFooter.DifferentFirst`

Set this property to **true** to specify a unique header or footer for the first page of a worksheet.

- `XIHeaderFooter.ScaleWithDoc`

Set this property to **true** to scale headers and footers proportionately when you scale a worksheet to fit your information on the specified number of pages.

- `XIHeaderFooter.AlignWithMargins`

Set this property to **true** to align the header and footer edges with page margins

Header and Footer Codes

The **Excel Export Library** supports specific codes that allows you to format the header/footer text and include dynamic information into a header or footer, such as a page number, current date and time, filename, worksheet name, and so on. Note that these codes can also be obtained as constant fields of the `XIHeaderFooter` class.

The supported header/footer codes are listed in the table below.

Code	XIHeaderFooter field	Description
&B	<code>XIHeaderFooter.Bold</code>	Turns bold on or off for the characters that follow.
&I	<code>XIHeaderFooter.Italic</code>	Turns italic on or off for the characters that follow.
&U	<code>XIHeaderFooter.Underline</code>	Turns underline on or off for the characters that follow.
&E	<code>XIHeaderFooter.DoubleUnderline</code>	Turns double underline on or off for the characters that follow.
&S	<code>XIHeaderFooter.Strikethrough</code>	Turns strikethrough on or off for the characters that follow.
&Y	<code>XIHeaderFooter.Subscript</code>	Turns subscript on or off for the characters that follow.
&X	<code>XIHeaderFooter.Superscript</code>	Turns superscript on or off for the characters that follow.
&L	<code>XIHeaderFooter.Left</code>	Left aligns the characters that follow.
&C	<code>XIHeaderFooter.Center</code>	Centers the characters that follow.

&R	XlHeaderFooter.Right	Right aligns the characters that follow.
&P	XlHeaderFooter.PageNumber	Inserts the current page number.
&N	XlHeaderFooter.PageTotal	Inserts the total number of pages in a workbook.
&D	XlHeaderFooter.Date	Inserts the current date.
&T	XlHeaderFooter.Time	Inserts the current time.
&Z	XlHeaderFooter.BookPath	Inserts the workbook file path.
&F	XlHeaderFooter.BookName	Inserts the name of a workbook file.
&A	XlHeaderFooter.SheetName	Inserts the name of a worksheet.
&"fontname"		Prints the characters that follow using the specified font. Be sure to enclose the font name in double quotation marks.
&nn		Prints the characters that follow using the specified font size. Use a two-digit number to specify the font size in points.
&P+number		Inserts the page number plus the specified number.
&P-number		Inserts the page number minus the specified number.
&&		Inserts an ampersand character.

Example**Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

The example below demonstrates how to specify different headers and footers for the odd-numbered and even-numbered pages of the printed worksheet.

C#

```
(PageViewAndLayoutActions.cs)
// Specify different headers and footers for the odd-numbered and even-numbered pages.
sheet.HeaderFooter.DifferentOddEven = true;
// Add the bold text to the header left section,
// and insert the workbook name into the header right section.
sheet.HeaderFooter.OddHeader = XlHeaderFooter.FromLCR(XlHeaderFooter.Bold + "Sample report", null, XlHeaderFooter.BookName);
// Insert the current page number into the footer right section.
sheet.HeaderFooter.OddFooter = XlHeaderFooter.FromLCR(null, null, XlHeaderFooter.PageNumber);
// Insert the workbook file path into the header left section,
// and add the worksheet name to the header right section.
sheet.HeaderFooter.EvenHeader = XlHeaderFooter.FromLCR(XlHeaderFooter.BookPath, null, XlHeaderFooter.SheetName);
// Insert the current page number into the footer left section
// and add the current date to the footer right section.
sheet.HeaderFooter.EvenFooter = XlHeaderFooter.FromLCR(XlHeaderFooter.PageNumber, null, XlHeaderFooter.Date);
```

Visual Basic

```
(PageViewAndLayoutActions.vb)
' Specify different headers and footers for the odd-numbered and even-numbered pages.
sheet.HeaderFooter.DifferentOddEven = True
' Add the bold text to the header left section,
' and insert the workbook name into the header right section.
sheet.HeaderFooter.OddHeader = XlHeaderFooter.FromLCR(XlHeaderFooter.Bold & "Sample report", Nothing, XlHeaderFooter.LeftSection)
' Insert the current page number into the footer right section.
sheet.HeaderFooter.OddFooter = XlHeaderFooter.FromLCR(Nothing, Nothing, XlHeaderFooter.RightSection)
' Insert the workbook file path into the header left section,
' and add the worksheet name to the header right section.
sheet.HeaderFooter.EvenHeader = XlHeaderFooter.FromLCR(XlHeaderFooter.BookPath, Nothing, XlHeaderFooter.LeftSection)
' Insert the current page number into the footer left section
' and add the current date to the footer right section.
sheet.HeaderFooter.EvenFooter = XlHeaderFooter.FromLCR(XlHeaderFooter.RightSection, Nothing, XlHeaderFooter.LeftSection)
```

How to: Insert Page Breaks in a Worksheet

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Printing](#) > [How to: Insert Page Breaks in a Worksheet](#)

The following example demonstrates how to insert a page break on a worksheet to start a new page. Page breaks in a worksheet are contained in two separate IXIPageBreaks collections: one for the *horizontal* (row) page breaks and one for the *vertical* (column) page breaks. To get access to these collections, use the IXISheet.RowPageBreaks and IXISheet.ColumnPageBreaks properties, respectively. To add a new page break, use the IXIPageBreaks.Add method and pass an index of the row or column after which a page break should be inserted. To remove a single page break from a worksheet, use the IXIPageBreaks.Remove or IXIPageBreaks.RemoveAt method. To clear all page breaks at once, use the IXIPageBreaks.Clear method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

(PageViewAndLayoutActions.cs)
// Insert a page break after the column "B".
sheet.ColumnPageBreaks.Add(2);
// Insert a page break after the tenth row.
sheet.RowPageBreaks.Add(10);

Visual Basic

(PageViewAndLayoutActions.vb)
' Insert a page break after the column "B".
sheet.ColumnPageBreaks.Add(2)
' Insert a page break after the tenth row.
sheet.RowPageBreaks.Add(10)

The image below illustrates the result (the workbook is opened in Microsoft® Excel®).

	A	B
1	Product	Sales
2	Camembert Pierrot	\$ 4,134.00
3	Gorgonzola Telino	\$ 3,196.00
4	Mascarpone Fabioli	\$ 3,795.00
5	Mozzarella di Giovanni	\$ 4,816.00
6	Gnocchi di nonna Alice	\$ 3,822.00
7	Gudbrandsdalsost	\$ 4,816.00
8	Gustaf's Knäckebröd	\$ 4,309.00
9	Queso Cabrales	\$ 4,590.00
10	Queso Manchego La Pastora	\$ 4,943.00
11	Raclette Courdavault	\$ 3,234.00
12	Singaporean Hokkien Fried Mee	\$ 4,426.00
13	Wimmers gute Semmelknödel	\$ 3,495.00
14		

Vertical Page Break

Horizontal Page Break

How to: Print Titles on a Worksheet

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Printing](#) > [How to: Print Titles on a Worksheet](#)

The example below demonstrates how to print specific rows and columns on every page. To do this, use methods of the XIPrintTitles object accessible from the IXISheet.PrintTitles property.

- To repeat specific rows at the top of the printed page, use the XIPrintTitles.SetRows method and pass the zero-based indexes of the first row and the last row to be printed.
- To repeat specific columns on the left side of the printed page, use the XIPrintTitles.SetColumns method and pass the zero-based indexes of the first column and the last column to be printed.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

(PageViewAndLayoutActions.cs)
// Print the first column and the first row on every page.
sheet.PrintTitles.SetColumns(0, 0);
sheet.PrintTitles.SetRows(0, 0);

Visual Basic

(PageViewAndLayoutActions.vb)
' Print the first column and the first row on every page.
sheet.PrintTitles.SetColumns(0, 0)
sheet.PrintTitles.SetRows(0, 0)

Data Validation

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Data Validation](#)

This section contains the following examples:

- [How to: Apply Data Validation](#)

How to: Apply Data Validation

[Office File API](#) > [Excel Export Library](#) > [Examples](#) > [Data Validation](#) > [How to: Apply Data Validation](#)

- The following example demonstrates how to manage the data validation rules.
1. Create a new `XIDataValidation` object that represents the new validation rule.
 2. Specify the cell range to which the validation rule is going to be applied. To do that, add the target range to the object's ranges collection, accessible through the `XIDataValidation.Ranges` property.
 3. Specify the type of data that is valid in the current rule by setting the `XIDataValidation.Type` property to the corresponding `XIDataValidationType` enumeration value.
 4. The operator for the validation rule can be specified using the `XIDataValidation.Operator` property.
 5. Specify the rule criterion by setting the `XIDataValidation.Criteria1` property. If the rule operator requires two criteria, specify the `XIDataValidation.Criteria2` property as well.
 6. To enable showing the error message, set the `XIDataValidation.ShowErrorMessage` to **true**. The `XIDataValidation.ErrorMessage`, `XIDataValidation.ErrorTitle` and `XIDataValidation.ErrorStyle` properties allows you to specify the error message text, title and style.
 7. You can also specify the input message that will be shown when the target cell is selected. To do that, set the `XIDataValidation.ShowInputMessage` to **true** and specify the text (`XIDataValidation.InputPrompt`) and the title (`XIDataValidation.PromptTitle`) of the message.
 8. Add the created validation rule to the corresponding worksheet collection. The collection can be accessed through the `IXISheet.DataValidations` property.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T253492>.

C#

```

(DataActions.cs)
// Apply data validation to cells.
// Restrict data entry in the range A2:A5 to a 5-digit number.
XlDataValidation validation = new XlDataValidation();
validation.Ranges.Add(XlCellRange.FromLTRB(0, 1, 0, 4));
validation.Type = XlDataValidationType.Custom;
validation.Criterial = "=AND(ISNUMBER(A2),LEN(A2)=5)";
// Add the specified rule to the worksheet collection of data validation rules.
sheet.DataValidations.Add(validation);
// Restrict data entry in the cell range C2:C5 to a whole number between 600 and 2000.
validation = new XlDataValidation();
validation.Ranges.Add(XlCellRange.FromLTRB(2, 1, 2, 4));
validation.Type = XlDataValidationType.Whole;
validation.Operator = XlDataValidationOperator.Between;
validation.Criterial1 = 600;
validation.Criteria2 = 2000;
// Display the error message.
validation.ErrorMessage = "The salary amount must be between 600$ and 2000$.";
validation.ErrorTitle = "Warning";
validation.ErrorStyle = XlDataValidationErrorStyle.Warning;
validation.ShowErrorMessage = true;
// Display the input message.
validation.InputPrompt = "Please enter a whole number between 600 and 2000";
validation.PromptTitle = "Salary";
validation.ShowInputMessage = true;
// Add the specified rule to the worksheet collection of data validation rules.
sheet.DataValidations.Add(validation);
// Restrict data entry in the cell range D2:D5 to a decimal number within the specified limits.
// Bonus cannot be greater than 10% of the salary.
validation = new XlDataValidation();
validation.Ranges.Add(XlCellRange.FromLTRB(3, 1, 3, 4));
validation.Type = XlDataValidationType.Decimal;
validation.Operator = XlDataValidationOperator.Between;
validation.Criterial1 = 0;
// Use a formula to specify the validation criterion.
validation.Criteria2 = "=C2*0.1";
// Display the error message.
validation.ErrorMessage = "Bonus cannot be greater than 10% of the salary.";
validation.ErrorTitle = "Information";
validation.ErrorStyle = XlDataValidationErrorStyle.Information;
validation.ShowErrorMessage = true;
// Add the specified rule to the worksheet collection of data validation rules.
sheet.DataValidations.Add(validation);
// Restrict data entry in the cell range E2:E5 to values in a drop-down list obtained from the cells G2:G5.
validation = new XlDataValidation();
validation.Ranges.Add(XlCellRange.FromLTRB(4, 1, 4, 4));
validation.Type = XlDataValidationType.List;
validation.Criterial1 = XlCellRange.FromLTRB(6, 1, 6, 4).AsAbsolute();
// Add the specified rule to the worksheet collection of data validation rules.
sheet.DataValidations.Add(validation);

```

Visual Basic


```
(DataActions.vb)
' Apply data validation to cells.
' Restrict data entry in the range A2:A5 to a 5-digit number.
Dim validation As New XlDataValidation()
validation.Ranges.Add(XlCellRange.FromLTRB(0, 1, 0, 4))
validation.Type = XlDataValidationType.Custom
validation.Criterial = "=AND(ISNUMBER(A2),LEN(A2)=5)"
' Add the specified rule to the worksheet collection of data validation ru
sheet.DataValidations.Add(validation)
' Restrict data entry in the cell range C2:C5 to a whole number between 60
validation = New XlDataValidation()
validation.Ranges.Add(XlCellRange.FromLTRB(2, 1, 2, 4))
validation.Type = XlDataValidationType.Whole
validation.Operator = XlDataValidationOperator.Between
validation.Criterial = 600
validation.Criteria2 = 2000
' Display the error message.
validation.ErrorMessage = "The salary amount must be between 600$ and 2000
validation.ErrorTitle = "Warning"
validation.ErrorStyle = XlDataValidationErrorStyle.Warning
validation.ShowErrorMessage = True
' Display the input message.
validation.InputPrompt = "Please enter a whole number between 600 and 2000
validation.PromptTitle = "Salary"
validation.ShowInputMessage = True
' Add the specified rule to the worksheet collection of data validation ru
sheet.DataValidations.Add(validation)
' Restrict data entry in the cell range D2:D5 to a decimal number within t
' Bonus cannot be greater than 10% of the salary.
validation = New XlDataValidation()
validation.Ranges.Add(XlCellRange.FromLTRB(3, 1, 3, 4))
validation.Type = XlDataValidationType.Decimal
validation.Operator = XlDataValidationOperator.Between
validation.Criterial = 0
' Use a formula to specify the validation criterion.
validation.Criteria2 = "=C2*0.1"
' Display the error message.
validation.ErrorMessage = "Bonus cannot be greater than 10% of the salary.
validation.ErrorTitle = "Information"
validation.ErrorStyle = XlDataValidationErrorStyle.Information
validation.ShowErrorMessage = True
' Add the specified rule to the worksheet collection of data validation ru
sheet.DataValidations.Add(validation)
' Restrict data entry in the cell range E2:E5 to values in a drop-down lis
validation = New XlDataValidation()
validation.Ranges.Add(XlCellRange.FromLTRB(4, 1, 4, 4))
validation.Type = XlDataValidationType.List
validation.Criterial = XlCellRange.FromLTRB(6, 1, 6, 4).AsAbsolute()
' Add the specified rule to the worksheet collection of data validation ru
sheet.DataValidations.Add(validation)
```

Snap Report API

[Office File API](#) > [Snap Report API](#)

The **Snap Report API** is the non-visual .NET library which allows you to create and customize mail-merge documents on the server, using the provided [API \(Application Programming Interface\)](#). Its main features are as follows.

- Create [mail merge](#) and tabular reports from scratch, and customize them in code.
- Format text and apply predefined or custom styles to report templates.
- Export to PDF, Microsoft Word formats (DOCX, DOC and RTF) and other popular formats (e.g., TXT, ODT and EPUB).
- Utilize charts, sparklines and bar codes.
- Support for RichEditDocumentServer features.
- Support for the medium trust permission level.

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this component or library in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

To learn how to implement a specific task for your reports using the Snap Report API, check the [Examples](#) section. To learn more about the Snap architecture, refer to the articles from the [Concepts](#) section.

Getting Started

[Office File API](#) > [Snap Report API](#) > [Getting Started](#)

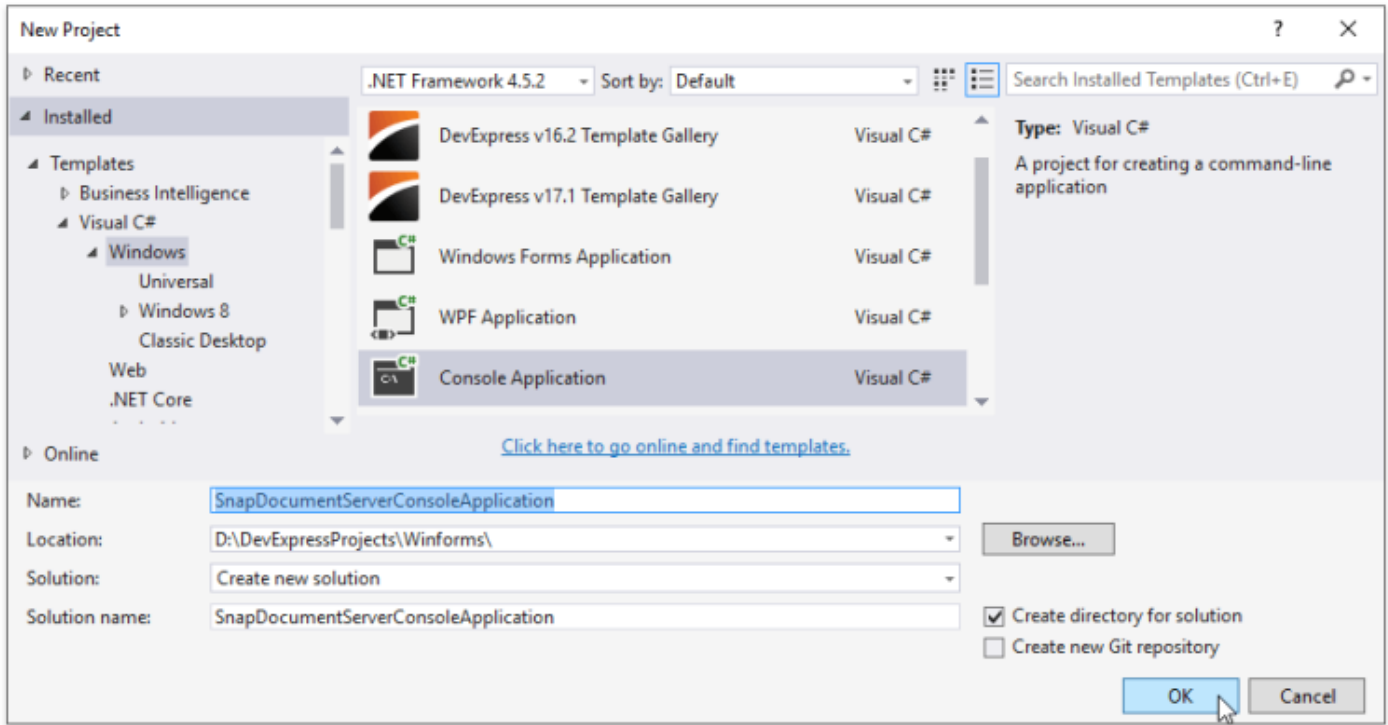
The following article describes how to get started with the [SnapDocumentServer](#) and create a simple report from scratch.

This tutorial consists of the following sections.

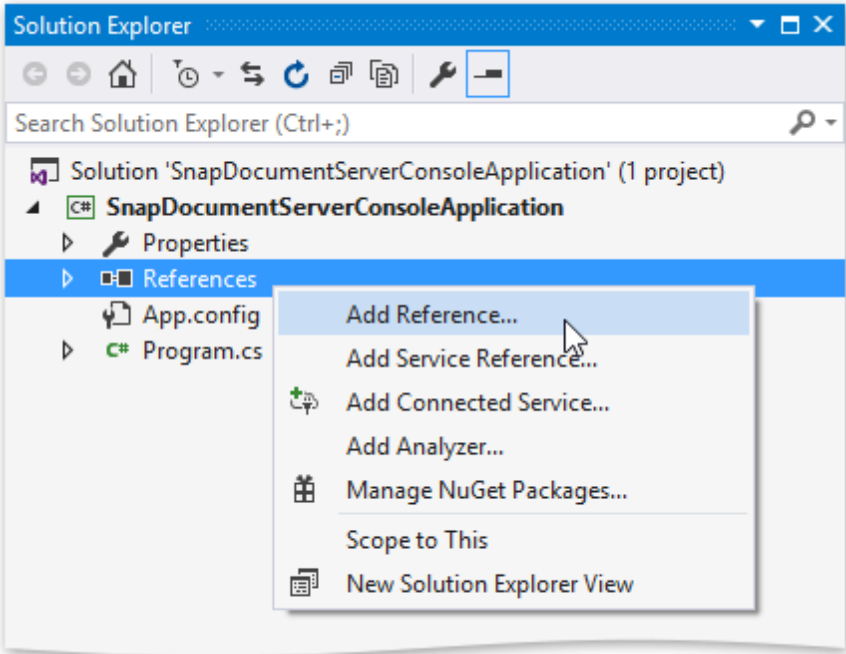
- [Create an Application](#)
- [Connect to Data](#)
- [Generate the Report Layout](#)

Create an Application

1. Start Visual Studio and create a new project by selecting **FILE | New | Project** in the main menu. In the invoked **New Project** dialog, select a **Console Application**, specify the project name and location, and click **OK**.



2. In the **Solution Explorer**, right-click the **References** node and select **Add Reference...** in the context menu.



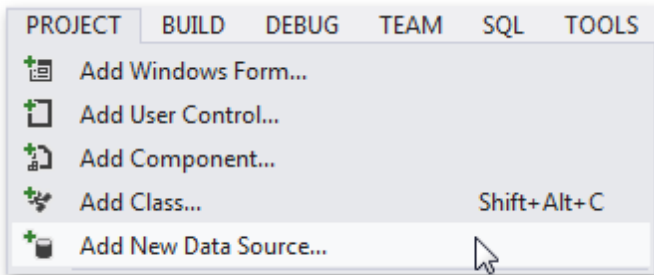
3. In the invoked **Reference Manager** dialog, add references to the following libraries.
- DevExpress.Data.v18.1.dll
 - DevExpress.Docs.v18.1.dll
 - DevExpress.Office.v18.1.Core.dll
 - DevExpress.RichEdit.v18.1.Core.dll
 - DevExpress.Snap.v18.1.Core.dll
4. Paste the following code in the **Main** method of the *Program.cs* file (**Main** procedure of the *Module1.vb* file for Visual Basic). The code snippet creates a new [SnapDocumentServer](#) instance, associates it with the data source, generates the tabular report and opens it in the default application registered for this file type.

Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T432062 .	
C#	
<pre>(Program.cs) //Create a new SnapDocumentServer instance. SnapDocumentServer server = new SnapDocumentServer(); SnapDocument document = server.Document; //Specify the server datasource. object datasource = GetDataSource(); document.DataSource = datasource; Console.WriteLine("Generating Document... "); // Generate the report. GenerateLayout(document); Console.WriteLine("Ok!"); Console.WriteLine("Press any key..."); Console.ReadKey(); document.ExportDocument("SnapDocumentServerTest.rtf", DocumentFormat.Rtf); System.Diagnostics.Process.Start("SnapDocumentServerTest.rtf");</pre>	
Visual Basic	

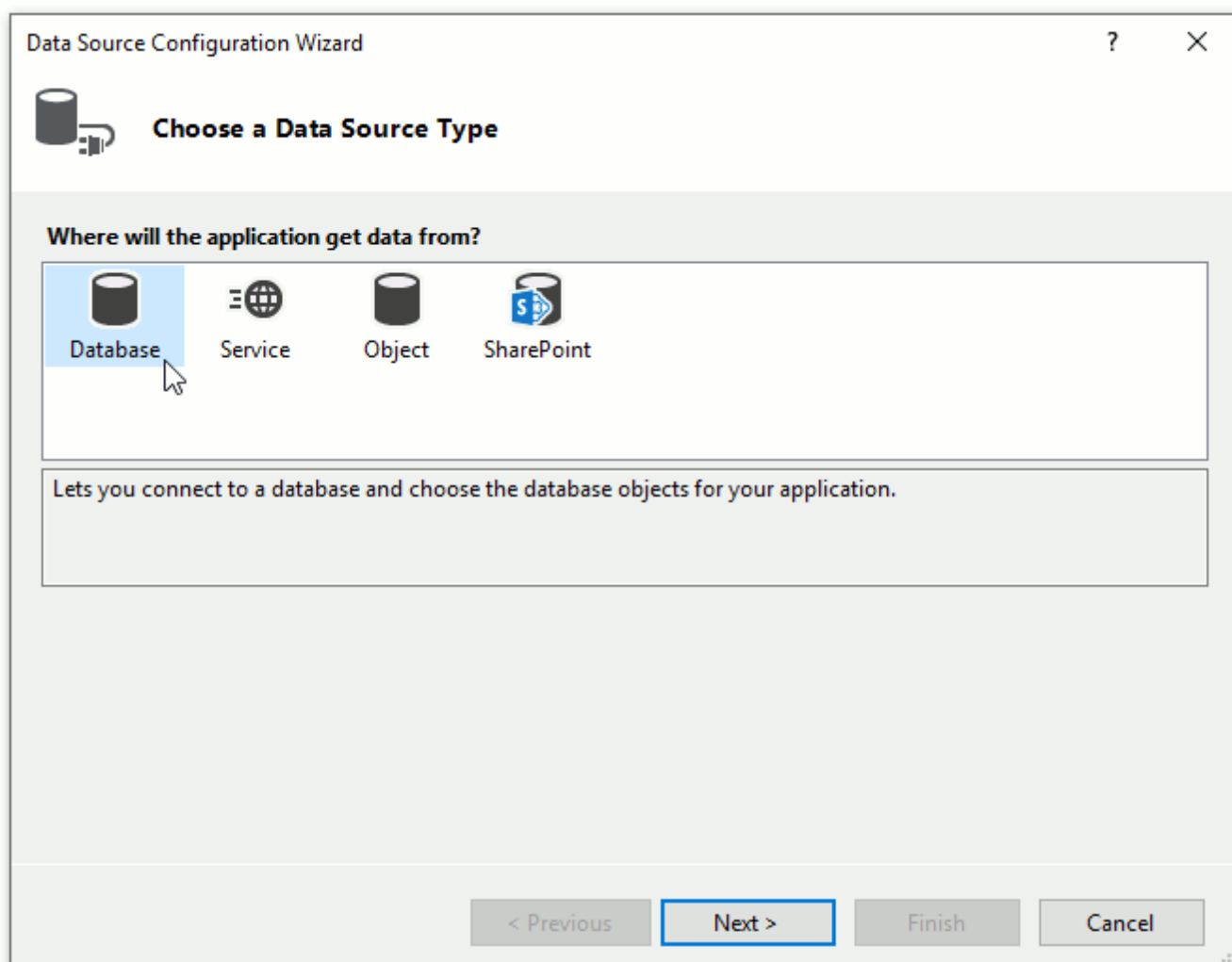
```
(Program.vb)
Dim server As New SnapDocumentServer()
Dim document As SnapDocument = server.Document
Dim datasource As Object = GetDataSource()
document.DataSource = datasource
Console.WriteLine("Generating Document... ")
GenerateLayout(document)
Console.WriteLine("Ok!")
Console.WriteLine("Press any key...")
Console.ReadKey()
System.Diagnostics.Process.Start("SnapDocumentServerTest.rtf")
```

Connect to Data

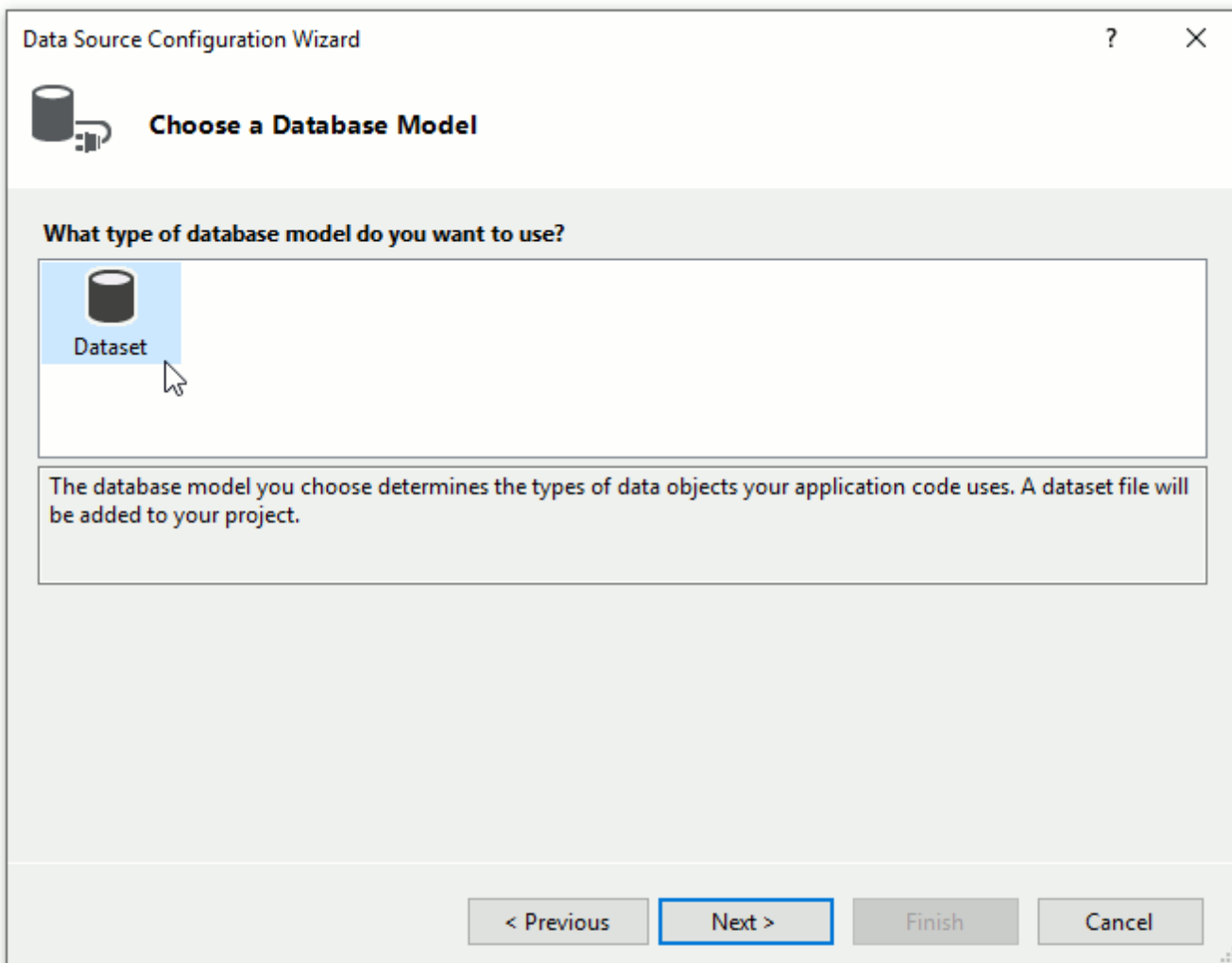
- Open the **PROJECT** menu and click the **Add New Data Source...** command.



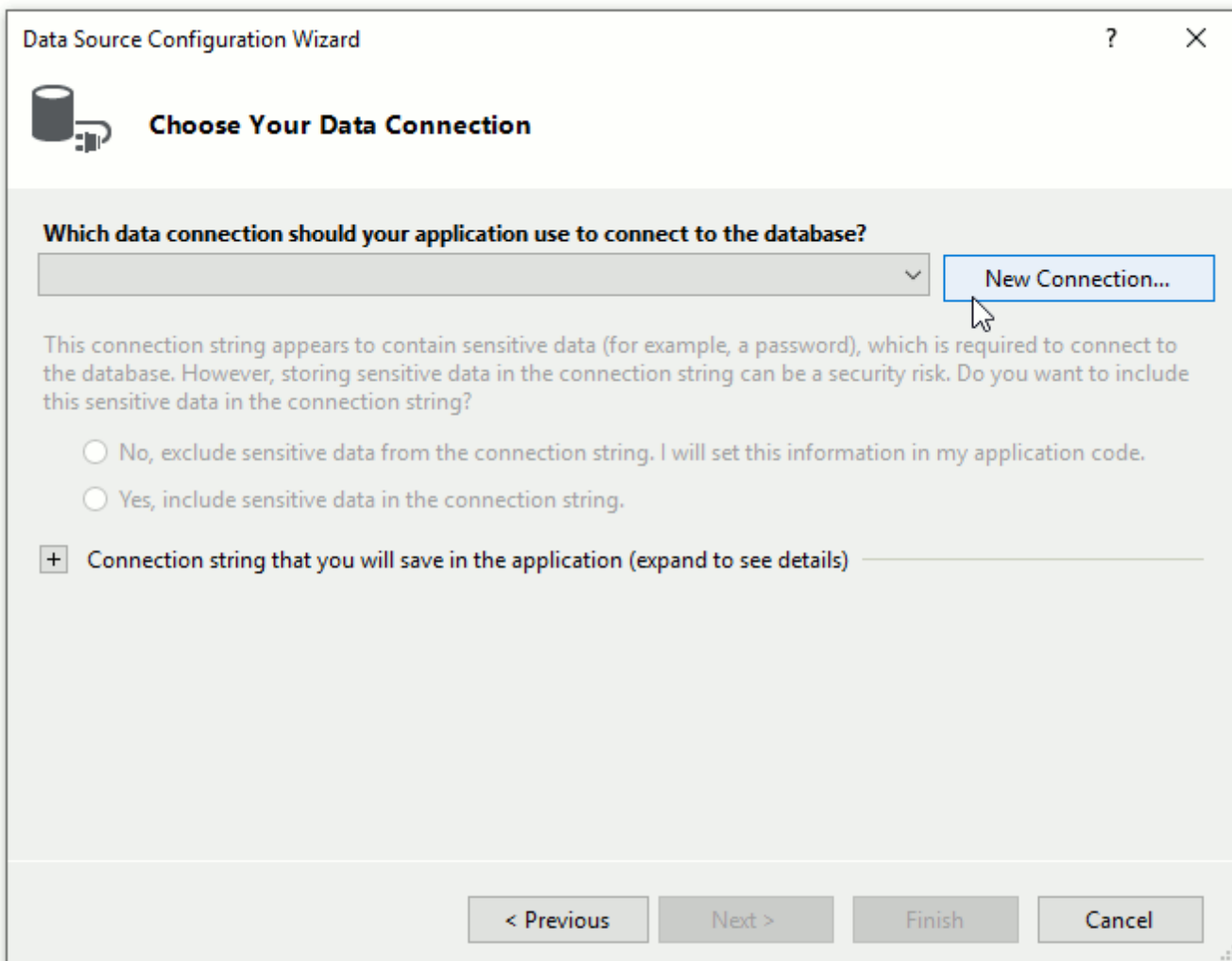
- In the invoked **Data Source Configuration Wizard**, select **Database** and click **Next**.



- Select **Dataset** to specify the type of a database model and click **Next**.



- On the next page, click **New Connection...** to specify which data connection should be used.



- In the invoked **Add Connection** dialog, set your data source to **Microsoft SQL Server Database File(SqlClient)** and specify the database file path. This tutorial uses a connection to the sample Northwind database (the **NWind.mdf** file is included in the product installation).

Add Connection ? X

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server Database File (SqlClient) Change...

Database file name (new or existing):
D:\DataBases\NWind.mdf Browse...

Log on to the server

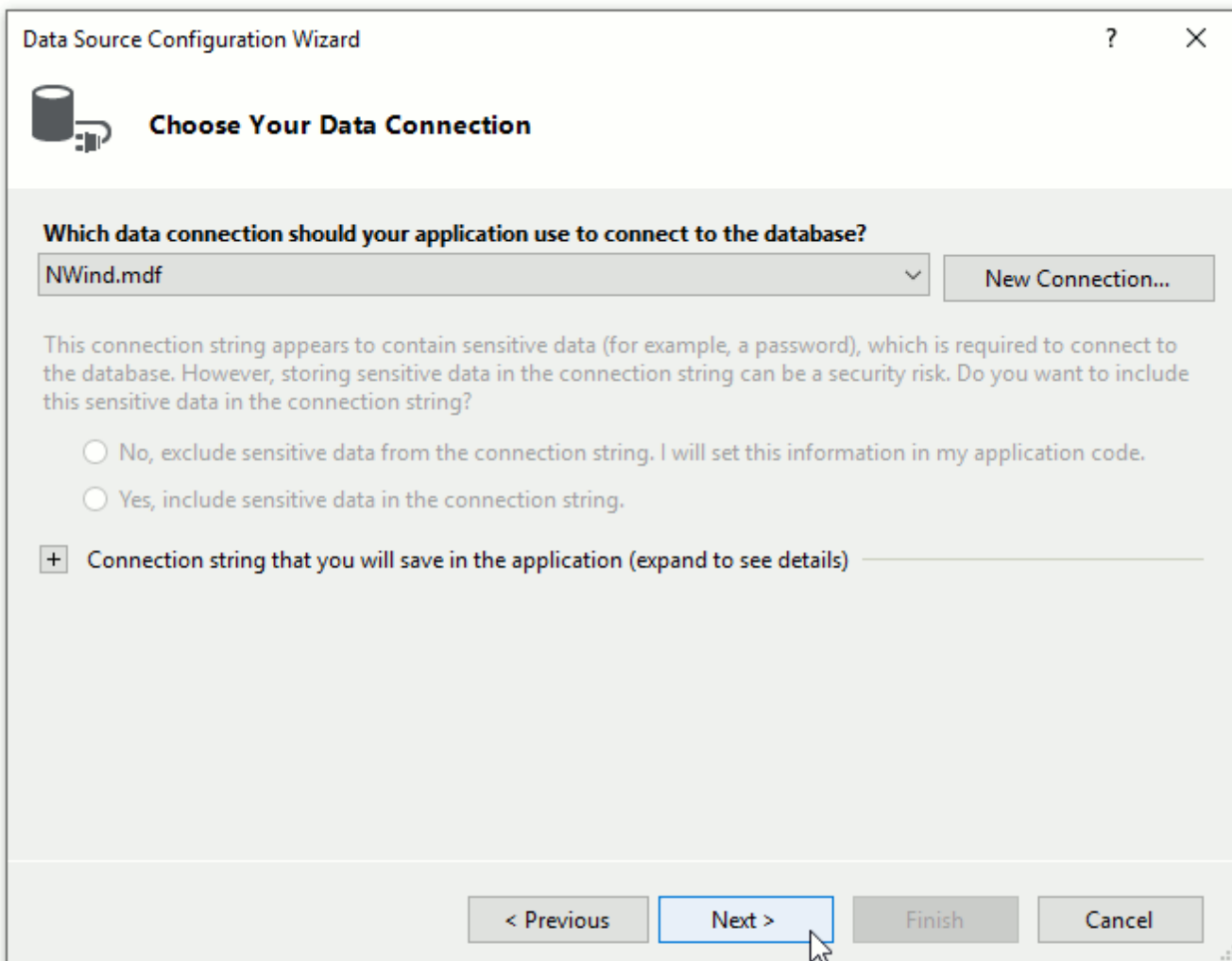
☒ Use Windows Authentication
☐ Use SQL Server Authentication

User name:
Password:
☐ Save my password

Advanced...

Test Connection OK Cancel

Click **OK** to apply the changes. In the **Data Source Configuration Wizard**, click **Next**.



The image shows a 'Data Source Configuration Wizard' window. The title bar says 'Data Source Configuration Wizard' with a question mark and a close button. Below the title bar is a header area with a database icon and the text 'Choose Your Data Connection'. The main area has a question: 'Which data connection should your application use to connect to the database?'. Below this is a dropdown menu showing 'NWind.mdf' and a 'New Connection...' button. A paragraph of text explains that the connection string might contain sensitive data like a password and asks if the user wants to include it. There are two radio buttons: 'No, exclude sensitive data from the connection string. I will set this information in my application code.' and 'Yes, include sensitive data in the connection string.'. Below the radio buttons is a checkbox labeled '+ Connection string that you will save in the application (expand to see details)'. At the bottom are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'. A mouse cursor is pointing at the 'Next >' button.

Data Source Configuration Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

NWind.mdf New Connection...

This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

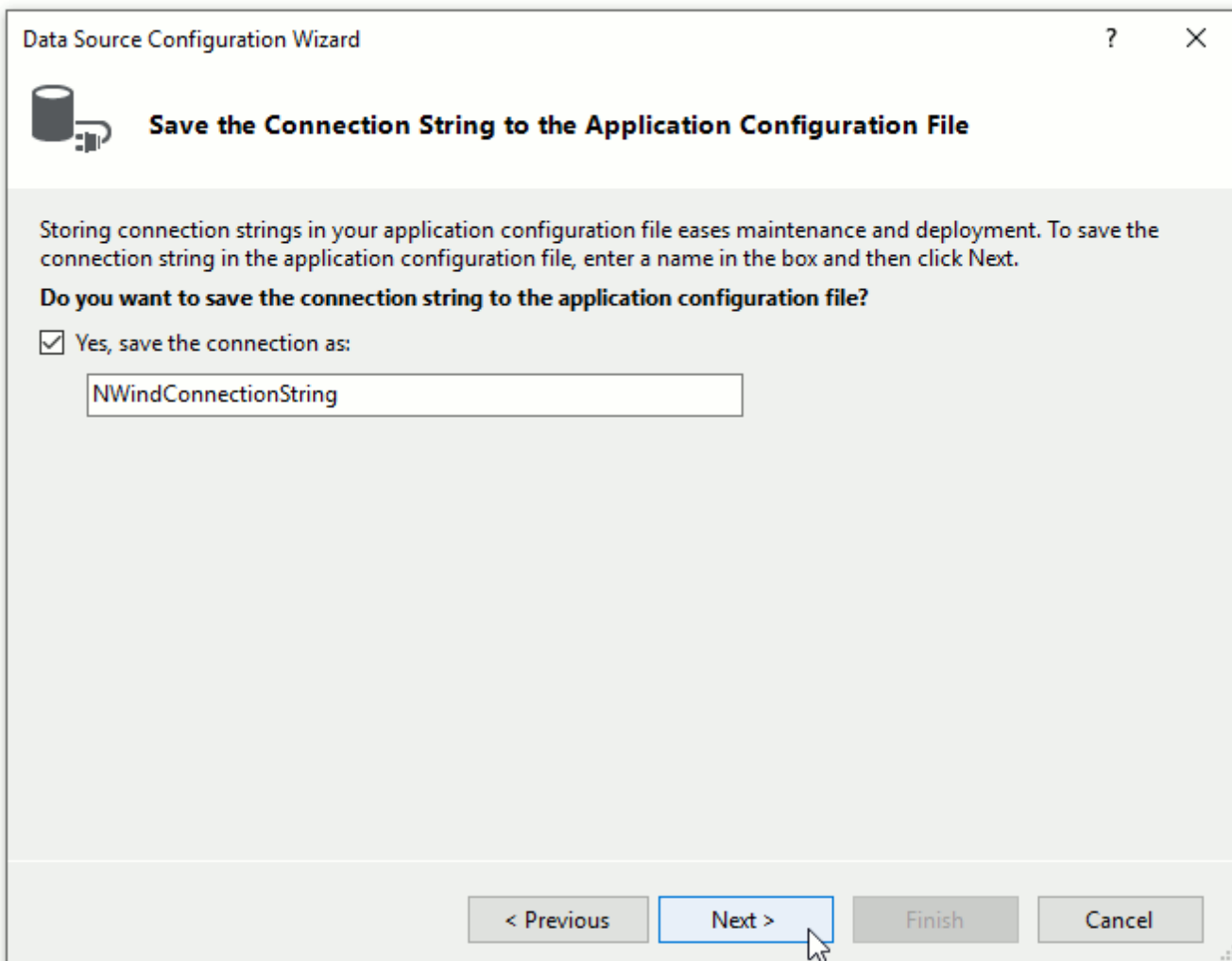
☐ No, exclude sensitive data from the connection string. I will set this information in my application code.

☐ Yes, include sensitive data in the connection string.

☐ + Connection string that you will save in the application (expand to see details)


< Previous Next > Finish Cancel

- Click **Next** on the following page to save the newly created connection string to the configuration file.



The image shows a 'Data Source Configuration Wizard' dialog box. At the top, there is a title bar with the text 'Data Source Configuration Wizard' and standard window controls. Below the title bar, there is a database icon and the heading 'Save the Connection String to the Application Configuration File'. The main text area contains instructions: 'Storing connection strings in your application configuration file eases maintenance and deployment. To save the connection string in the application configuration file, enter a name in the box and then click Next.' This is followed by the question 'Do you want to save the connection string to the application configuration file?'. Below this question is a checked checkbox labeled 'Yes, save the connection as:'. Underneath the checkbox is a text input field containing the text 'NWindConnectionString'. At the bottom of the dialog, there are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border and a mouse cursor is pointing at it.

Data Source Configuration Wizard

 **Save the Connection String to the Application Configuration File**

Storing connection strings in your application configuration file eases maintenance and deployment. To save the connection string in the application configuration file, enter a name in the box and then click Next.

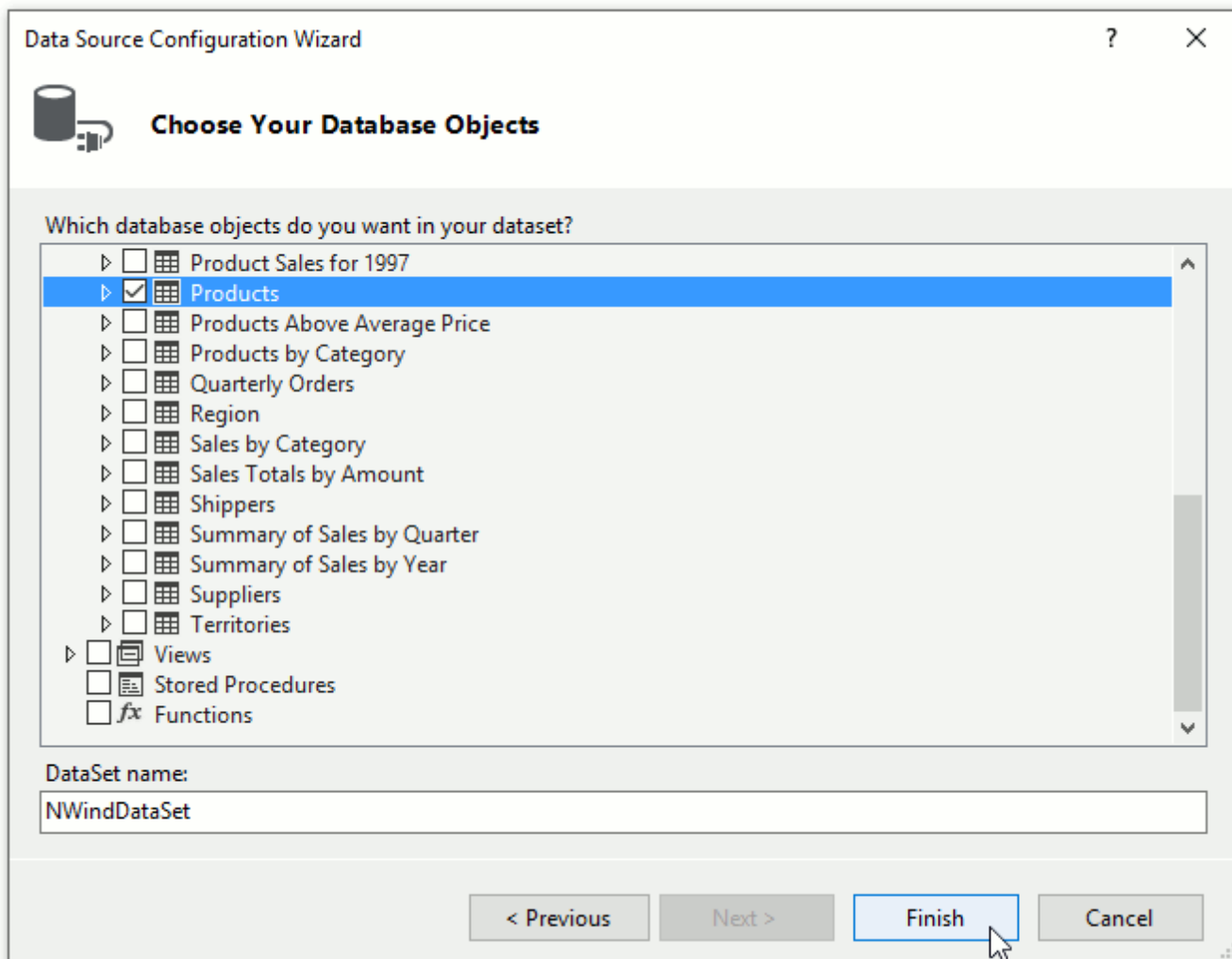
Do you want to save the connection string to the application configuration file?

☒ Yes, save the connection as:

NWindConnectionString

< Previous Next > Finish Cancel

- The final wizard page allows you to choose which tables to obtain from the database. After this step, click **Finish**.



- Create a new DataSet object and populate it with data from the database by adding the following code.

◉ Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=T432062>.

C#

```
(Program.cs)
private static object GetDataSource()
{
    NWindDataSet dataSource = new NWindDataSet();
    var connection = new SqlConnection();
    connection.ConnectionString = Properties.Settings.Default.NWindConnect
    ProductsTableAdapter products = new ProductsTableAdapter();
    products.Connection = connection;
    products.Fill(dataSource.Products);
    return dataSource.Products;
}
```

Visual Basic

```

(Program.vb)
Private Shared Function GetDataSource() As Object
    Dim dataSource As New NWindDataSet()
    Dim connection = New SqlConnection()
    connection.ConnectionString = My.Settings.Default.NWindConnectionString
    Dim products As New ProductsTableAdapter()
    products.Connection = connection
    products.Fill(dataSource.Products)
    Return dataSource.Products
End Function

```

Generate a Report Layout

- Use the code below to generate a simple template and fill it with data. The code snippet created a snap list with three data entries and a header, and exports the result to the *SnapDocumentServerTest.rtf* file.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T432062>.

C#

```

(Program.cs)
static void GenerateLayout(SnapDocument document)
{
    // Add a Snap list to the document.
    SnapList list = document.CreateSnList(document.Range.End, @"List");
    list.BeginUpdate();
    list.EditorRowLimit = 100500;
    // Add a header to the Snap list.
    SnapDocument listHeader = list.ListHeader;
    Table listHeaderTable = listHeader.Tables.Create(listHeader.Range.End,
    TableCellCollection listHeaderCells = listHeaderTable.FirstRow.Cells;
    listHeader.InsertText(listHeaderCells[0].ContentRange.End, "Product Na
    listHeader.InsertText(listHeaderCells[1].ContentRange.End, "Units in S
    listHeader.InsertText(listHeaderCells[2].ContentRange.End, "Unit Price
    //Create the row template and fill it with data.
    SnapDocument listRow = list.RowTemplate;
    Table listRowTable = listRow.Tables.Create(listRow.Range.End, 1, 3);
    TableCellCollection listRowCells = listRowTable.FirstRow.Cells;
    listRow.CreateSnText(listRowCells[0].ContentRange.End, @"ProductName")
    listRow.CreateSnText(listRowCells[1].ContentRange.End, @"UnitsInStock")
    listRow.CreateSnText(listRowCells[2].ContentRange.End, @"UnitPrice \$
    list.EndUpdate();
    list.Field.Update();
}

```

Visual Basic

```
(Program.vb)
Private Shared Sub GenerateLayout(ByVal document As SnapDocument)
    ' Add a Snap list to the document.
    Dim list As SnapList = document.CreateSnList(document.Range.End, "List")
    list.BeginUpdate()
    list.EditorRowLimit = 100500
    ' Add a header to the Snap list.
    Dim listHeader As SnapDocument = list.ListHeader
    Dim listHeaderTable As Table = listHeader.Tables.Create(listHeader.Range.End, 1)
    Dim listHeaderCells As TableCellCollection = listHeaderTable.FirstRow.Cells
    listHeader.InsertText(listHeaderCells(0).ContentRange.End, "Product Name")
    listHeader.InsertText(listHeaderCells(1).ContentRange.End, "Units in Stock")
    listHeader.InsertText(listHeaderCells(2).ContentRange.End, "Unit Price")
    ' Create the row template and fill it with data
    Dim listRow As SnapDocument = list.RowTemplate
    Dim listRowTable As Table = listRow.Tables.Create(listRow.Range.End, 1)
    Dim listRowCells As TableCellCollection = listRowTable.FirstRow.Cells
    listRow.CreateSnText(listRowCells(0).ContentRange.End, "ProductName")
    listRow.CreateSnText(listRowCells(1).ContentRange.End, "UnitsInStock")
    listRow.CreateSnText(listRowCells(2).ContentRange.End, "UnitPrice \ $ $")
    list.EndUpdate()
    list.Field.Update()
    document.ExportDocument("SnapDocumentServerTest.rtf", DocumentFormat.Rtf)
End Sub
```

 **Tip**

This example demonstrates how to generate the template in code. You can also use a document template created in the DevExpress Snap at runtime. To learn how to generate a template using the Snap UI, check the Snap List and Document Template topic.

The following image shows the report generated after executing the code above (the document is opened in Microsoft Word).

Product Name	Units in Stock	Unit Price
Chai	39	\$18.00
Chang	17	\$19.00
Aniseed Syrup	13	\$10.00
Chef Anton's Cajun Seasoning	53	\$22.00
Chef Anton's Gumbo Mix	0	\$21.35
Grandma's Boysenberry Spread	120	\$25.00
Uncle Bob's Organic Dried Pears	15	\$30.00
Northwoods Cranberry Sauce	6	\$40.00
Mishi Kobe Niku	29	\$97.00
Ikura	31	\$31.00
Queso Cabrales	22	\$21.00
Queso Manchego La Pastora	86	\$38.00
Konbu	24	\$6.00
Tofu	35	\$23.25
Genen Shouyu	39	\$15.50
Pavlova	29	\$17.45
Alice Mutton	0	\$39.00
Carnarvon Tigers	42	\$62.50
Teatime Chocolate Biscuits	25	\$9.20
Sir Rodney's Marmalade	40	\$81.00

Note

This tutorial concentrates on a very limited set of actions that can be performed with Snap. For more information on the Snap reporting functionality, see the tutorials in the [Examples](#) section.

See Also
[Examples](#)

Concepts

[Office File API](#) > [Snap Report API](#) > [Concepts](#)

The topics in this section give you basic information on the architecture and functionality of the **Snap Document Server**.

- [Application Programming Interface](#)
- [Snap Fields](#)
- [File Formats](#)

Application Programming Interface

[Office File API](#) > [Snap Report API](#) > [Concepts](#) > [Application Programming Interface](#)

This topic provides a software developer's perspective on the architecture of Snap Report API, including its object model overview.

The topic includes the following sections.

- [Snap Report Application Programming Interface](#)
- [Runtime Customization](#)

Snap Report Application Programming Interface

To create documents in code, Snap Report API includes the following main elements.

1. SnapDocument provides methods, properties and events to manage a document's content, data sources and parameters.
2. SnapList is an area of a Snap document, which uses templates to define the layout and formatting of data. These templates have a basic tabular structure that follows that of the Table class. In the created document, these templates are serialized as [field codes](#) that use a special markup.

To provide data shaping capabilities, the Snap list exposes the following properties:

SnapList.Filters;
SnapList.Groups;
SnapList.Sorting.

3. The following elements provide the functionality required to present data in Snap documents:

SnapBarCode;
SnapCheckBox;
SnapHyperlink;
SnapImage;
SnapSparkline;
SnapText.

Refer to the [Examples](#) section for code examples illustrating the programmatic generation of a document's layout using the above API elements.

Runtime Customization

Each [Snap Field](#) in a document corresponds to a specific SnapEntity that is returned by the ISnapFieldOwner.ParseField method. Any modification of an entity at runtime must be wrapped in the SnapEntity.BeginUpdate and SnapEntity.EndUpdate method calls.

After calling the SnapEntity.BeginUpdate method, the SnapEntity.Active property becomes enabled until the SnapEntity.EndUpdate method is called. A document can only contain one active entity at one time.

The following requirements apply to the proper customization of a Snap entity at runtime.

- To learn whether or not the customization is opened for an entity, consult its SnapEntity.Active property value.

To learn whether or not the customization is opened for a specific entity, use the ISnapFieldOwner.ActiveEntity property.

- To open an entity for runtime customization, call its SnapEntity.BeginUpdate method.

To apply the new settings, call the SnapEntity.EndUpdate method.

Do not call the SnapEntity.EndUpdate method for a parent entity until there is no active nested entity left within it.

Nested Fields

If a document contains multiple entities that are nested inside each other (e.g., in case of a [master-detail report](#)), only the outermost (parent) field in this document is active.

The parent field's data is contained in one or more SnapDocument objects assigned to the field templates (e.g., SnapList.RowTemplate, SnapListGroupInfo.Header, or SnapListGroupInfo.Footer).

To modify the content of a nested field within a parent field's document, do the following.

1. Obtain the nested field from a corresponding SnapDocument.
2. Parse the field by calling the ISnapFieldOwner.ParseField method that returns the appropriate SnapEntity.
3. Call the SnapEntity.BeginUpdate method of the returned entity.
4. After making the required changes, make sure to call the SnapEntity.EndUpdate method for the nested entity.

5. Call the SnapEntity.EndUpdate method for the parent entity.

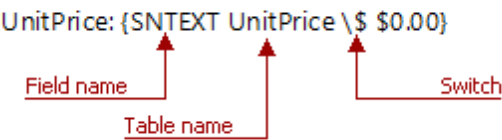
Snap Fields

[Office File API](#) > [Snap Report API](#) > [Concepts](#) > [Snap Fields](#)

This topic describes the implementation of fields in the context of a **Snap Report API**.

The Fields are defined as a set of codes that instructs the program to insert text and graphics into a document automatically. It means that Snap fields serve as placeholders for dynamic data. The data inserted into a document when the application processes a field's codes is called the field result for that field.

Field codes appear between curly braces ({ }).



As you can see, the field code consists of the **Field name** which points what specified dynamic data should be inserted, the **table name** the data should be taken from and the **switch** - a markup element specifying an optional setting available for a particular field.

Fields are divided into two types, based on their intended use.

- [Snap Fields](#)
- [Document Fields](#)

Snap Fields

Snap fields are used as placeholders for data-bound content.

The following table lists the field codes that are showcased by Snap to insert dynamic data into a document.

Field Name	Functionality	Switches
SNLIST	Inserts the specified list.	<ul style="list-style-type: none">• \name - specifies the name of a Snap list.• \t - specifies the template used to display every data row of a data source.• \listheader - specifies the content of the list header.• \listfooter - specifies the content of the list footer.• \erl - specifies the maximum number of data rows to include in the document during the design session. The default is 20.• \separator - specifies the separator to be inserted between data rows of a Snap list.• \tga, \tgb, \tgc ... - specify the content of the Snap list's group headers and group footers.
SNTTEXT	Inserts the specified text.	<ul style="list-style-type: none">• \ \$ - specifies the format string for the displayed content of the field.• \sinv - for summary fields, specifies whether or not to ignore null values when calculating a summary function.• \sr - specifies the report area for which the summary function is

		<p>calculated (summary running).</p> <p>The following modes are available:</p> <ul style="list-style-type: none"> - Group - the function is calculated for every group; - None - the function is not calculated; - Page - the function is calculated for every page; - Report - the function is calculated for the entire report. <ul style="list-style-type: none"> • \sf - specifies the summary function to be calculated. The result of the summary function will be displayed as the value of a Snap field. • \b - specifies the text before the content obtained from the bound data field (if this content is not empty). • \f - specifies the text after the content obtained from the bound data field (if this content is not empty). • \klp - specifies whether to keep last paragraph, that is to insert the current paragraph spacing after its last occurrence in the document. • \tf - specifies the text format, i.e. the format of the text content - rtf, html etc.
SNCHECKBOX	Inserts the specified check box.	<ul style="list-style-type: none"> • \c - specifies the state of a check box. The following states are available: <ul style="list-style-type: none"> - c - checked; - u - unchecked; - i - indeterminate.
SNCHART	Inserts the specified chart.	The switches of this Snap field may contain complex/binary data, and are not intended to be manually inserted into markup.
SNBARCODE	Inserts the specified bar code.	The switches of this Snap field may contain complex/binary data, and are not intended to be manually inserted into markup.
SNIMAGE	Inserts an image with a specified height and width.	<ul style="list-style-type: none"> • \k - specifies whether to preserve the image box size or the ratio of the original image. The following values are available: <ul style="list-style-type: none"> - size; - scale. • \s - specifies the size mode for an image displayed by a field. • \sx - specifies the width of the image box. • \sy - specifies the height of the image box.
SNINDEX	Inserts the specified row index.	<ul style="list-style-type: none"> • \\$ - specifies the format string for the displayed content of the field.

		<ul style="list-style-type: none"> • \im - specifies the group mode of a row index. The following group modes are available: <ul style="list-style-type: none"> - Restarted - row index numbers restart for each group in a grouped Snap list; - Continued - row index numbers continue for each group in a grouped Snap list.
SNSPARKLINE	Inserts the specified sparkline.	<ul style="list-style-type: none"> • \dsn - specifies the data member name for hierarchical data sources. • \w - specifies the width of a sparkline. • \h - specifies the height of a sparkline. • \vt - specifies the view type of a sparkline. • \hmxp - specifies whether or not to highlight the sparkline point with the highest value among all points. • \hmnp - specifies whether or not to highlight the sparkline point with the lowest value among all points. • \hsp - specifies whether or not to highlight the start point of a sparkline. • \hep - specifies whether or not to highlight the end point of a sparkline. • \cl - specifies the color used to draw a sparkline. • \mxpcl - specifies the color used to draw the sparkline point with the highest value among all data points. • \mnpcl - specifies the color used to draw the sparkline point with the lowest value among all data points. • \spcl - specifies the color used to draw the start point of a sparkline. • \epcl - specifies the color used to draw the end point of a sparkline. • \npcl - specifies the color used to draw sparkline points that have negative values. • \lnw - specifies the width of the chart line for the Line and Area sparkline view types. • \hnp - specifies whether or not to highlight all negative points in the Line, Area and Bar sparkline view types. • \sm - specifies whether or not to show point markers in the Line and Area sparkline view types. • \msz - specifies the size of markers for data points in the Line and Area sparkline view types. • \mxpmzs - specifies the marker size of the data point with the highest value among all data points in the Line and Area sparkline view types.

		<ul style="list-style-type: none"> • \mnpmsz - specifies the marker size of the data point with the lowest value among all data points in the Line and Area sparkline view types. • \spmsz - specifies the size of the start point's marker in the Line and Area sparkline view types. • \epmsz - specifies the size of the end point's marker in the Line and Area sparkline view types. • \npmsz - specifies the marker size of all data points that have negative values in the Line and Area sparkline view types. • \mcl - specifies the color of line markers for the Line and Area sparkline view types. • \ao - specifies the opacity (0-255) of the area fill color for the Area sparkline view type. • \bd - specifies the distance between two bars of a sparkline of the Bar or WinLoss view type.
SNHYPERLINK	Inserts the specified hyperlink.	<ul style="list-style-type: none"> • \d - specifies the data member containing the hyperlink display text. • \o - specifies the text of the tooltip that appears when the mouse cursor points to a hyperlink. • \t - specifies the target frame of a link.

Document Fields

Document fields are used as placeholders for other types of dynamically generated content such as current date and time, a page number, or a table of contents.

Snap inherits most of the field codes described in this section from its ancestor RichEditControl, to ensure that they are supported in Snap documents.

The following table describes the most commonly used fields.

Field Name	Functionality	Switches
CREATEDATE	Inserts the current date and time. After a mail merge, the field is replaced with the date and time of the mail merge operation.	None
DATE	Inserts the current date and time.	None
TIME	Inserts the current time.	None
HYPERLINK	Enables you to navigate to another location or to a bookmark.	None
IF	Compares two values, and inserts text according to the results of the comparison.	None
INCLUDEPICTURE	Inserts the specified picture.	<ul style="list-style-type: none"> • \d - this switch prevents you from storing the image in a document file.

		For example, the {INCLUDEPICTURE "C:\TMP\My picture.png" \d} field retains a link to the picture, but the picture itself is not included in the saved document.
NUMPAGES	Inserts the total number of pages.	None
PAGE	Inserts the number of the page containing the field.	None
SEQ	Provides sequential numbering in the document.	<ul style="list-style-type: none"> • \c - repeats the closest preceding sequence number. • \h - hides the field result. • \n - inserts the next sequence number for the specified item. This mode is used by default. • \r - resets the sequence number to the specified integer number.
TC	Defines entries for the table of contents.	<ul style="list-style-type: none"> • \f - specifies the type of items collected in a particular content list. Use a unique type identifier for each type of list. This switch is required for including an entry into the corresponding table of contents. • \l - the level of the TC entry. If no level is specified, level 1 is used.
TOC	Builds a table of contents. Without switches, the table of contents is built upon styles with set outline levels. Outline levels assigned to individual paragraphs are disregarded unless the \u switch is specified.	<ul style="list-style-type: none"> • \b - only includes entries from the portion of the document marked by the bookmark named by text specified in a field argument. • \h - inserts table of contents entries as hyperlinks. • \n - omits page numbers from the table of contents. Page numbers are omitted from all levels unless a range of entry levels is specified. For example, { TOC \n 1-1 } omits page numbers from level 1. • \u - uses the applied paragraph outline level. • \o - builds a table of contents from paragraphs with specified outline levels. For example, { TOC \o "1-3" } only lists paragraphs with outline levels 1 through 3. • \t - builds a table of contents from paragraphs formatted with specified styles. For example, {TOC \t"Title,1,subtitle,2,customtitle,3"} builds a table of contents from paragraphs formatted with the styles "Title", "subtitle" and "customtitle". The number after each style name indicates the entry level corresponding to that style. • \c - builds a table of contents from items that are numbered by an SEQ field. The sequence identifier designated by text in this switch's field-argument shall match the

		<p>identifier in the corresponding SEQ field.</p> <ul style="list-style-type: none">• \s - for entries numbered with an SEQ field, adds a prefix to the page number. The prefix depends on the type of entry.• \d - when used with \s, defines the separator between sequence and page numbers. The default separator is a hyphen (-).• \f - builds a table of contents from TC fields. The TC field identifier must match the text exactly in this switch's field-argument.• \l - used with the \f key. Includes TC fields that assign entries to one of the specified levels.
--	--	--

See Also

- General Format Switch
- Numeric Format Switch
- Date and Time Format Switch
- Rich Text Field Codes

File Formats

[Office File API](#) > [Snap Report API](#) > [Concepts](#) > [File Formats](#)

The file formats supported by Snap Report API are divided into the following categories.

- [Native Document Format](#)
- [Document Formats for Import](#)
- [Document Formats for Export](#)

Native Document Format (SNX)

The Snap native document format (**.SNX**) is a report template which stores the layout without the actual data.

Do not save a report template in any other format if you do not intend to drop Snap-specific features. When a document is saved in format other than SNX, Snap fields are replaced with their values and data connection information is removed.

Supported Document Formats for Import

Report templates can be built upon the document loaded by Snap Report API.

A SnapControl can load a document in the following file format.

- **DOC** (Microsoft® Word® 97 - 2003 document);
- **DOCX** (Office® Open XML document);
- **HTML** (HyperText Markup Language);
- **MHTML** / **MHT** (Web archive, single file);
- **RTF** (Rich Text Format);
- **TXT** (Plain text);
- **ODT** (OpenDocument text document);
- **XML** (Microsoft® Word® XML document);

Supported Document Formats for Export

The SnapDocumentServer creates a report based on the loaded report template. Subsequently the report can be saved in different file formats.

A Snap report can be exported to the following file formats.

- **DOC** (Microsoft® Word® 97 - 2003 document).
- **DOCX** (Office® Open XML document);
- **HTML** (HyperText Markup Language);
- **MHTML** / **MHT** (Web archive, single file);
- **PDF** (Portable Document Format);
- **RTF** (Rich Text Format);
- **TXT** (Plain text);
- **ODT** (OpenDocument text format);
- **XML** (Microsoft® Word® XML document);
- **Image** (BMP, EMF, WMF, GIF, JPEG, PNG or TIFF format).

Examples

[Office File API](#) > [Snap Report API](#) > [Examples](#)

This section provides a full list of examples (grouped by features) contained in this help.

Data Shaping

- [How to: Insert Data](#)
- [How to: Format Data](#)
- [How to: Group Data](#)
- [How to: Sort Data](#)
- [How to: Filter Data](#)

Data Visualization Tools

- [How to: Create Parameter](#)
- [How to: Create Check Box](#)
- [How to: Create Barcode](#)
- [How to: Create Sparkline](#)
- [How to: Create Calculated Field](#)

Mail Merge

- [How to: Generate Master-Detail Mail Merge Documents](#)

How to: Insert Data

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Insert Data](#)

The following example describes how to add data to a document in the Snap Report API.

The Snap document has a basic tabular structure. When data is added at runtime, data fields are placed to the automatically created table cells. To add new data to the document in the API, you need to insert data fields to the table cells as well. To complete this task, follow the steps below:

1. Load the document template using the RichEditDocumentServer.LoadDocumentTemplate method.
2. Call the ISnapFieldOwner.CreateSnList method to create a new Snap list or retrieve one of the existing lists by calling the ISnapFieldOwner.FindListByName method.
3. Convert the document fields to Snap Fields. To do that, call the ISnapFieldOwner.ParseField method. Note that if this step is skipped, you will not be able to modify the document.
4. Enable modification of the list by calling the SnapEntity.BeginUpdate method.
5. Access the row template through the SnapList.RowTemplate property.
6. To add a new cell in the row, call the TableCellCollection.InsertAfter or TableCellCollection.InsertBefore method, depending on where you want to add the cell.
7. To retrieve one of the existing document table cells, access the cell collection using the TableRow.Cells property and get the desired cell by its index.
8. Insert the required data field. To do that, use the ISnapFieldOwner.CreateSnText method. Use the position within the desired table cell as a method parameter. The TableCell.ContentRange property provides access to the cell range.
9. If necessary, create a cell in the list header. To do that, access the list header through the SnapList.ListHeader property, retrieve the header's table row and use the SubDocument.InsertText method to insert the desired content to the cell. To create a new table cell, use the TableCellCollection.InsertAfter or TableCellCollection.InsertBefore method.
10. Update the list's fields by calling the FieldCollection.Update method. Updating the fields activates the modifications you have created.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataShapingActions.cs)
server.LoadDocumentTemplate("Template.snx");
SnapList list = server.Document.FindListByName("Data Source 11");
server.Document.ParseField(list.Field);
list.BeginUpdate();
// Add a header to the Snap list.
SnapDocument listHeader = list.ListHeader;
Table listHeaderTable = listHeader.Tables[0];
TableCellCollection listHeaderCells = listHeaderTable.FirstRow.Cells;
listHeader.InsertText(listHeaderCells.InsertAfter(2).ContentRange.End, "Quantity per Unit");
// Add data to the row template:
SnapDocument listRow = list.RowTemplate;
Table listRowTable = listRow.Tables[0];
TableCellCollection listRowCells = listRowTable.FirstRow.Cells;
listRow.CreateSnText(listRowCells.InsertAfter(2).ContentRange.End, @"QuantityPerUnit");
list.EndUpdate();
list.Field.Update();
```

Visual Basic

```
(DataShapingActions.vb)
server.LoadDocumentTemplate("Template.snx")
Dim list As SnapList = server.Document.FindListByName("Data Source 11")
server.Document.ParseField(list.Field)
list.BeginUpdate()
' Add a header to the Snap list.
Dim listHeader As SnapDocument = list.ListHeader
Dim listHeaderTable As Table = listHeader.Tables(0)
Dim listHeaderCells As TableCellCollection = listHeaderTable.FirstRow.Cells
listHeader.InsertText(listHeaderCells.InsertAfter(2).ContentRange.End, "Qu
' Add data to the row template:
Dim listRow As SnapDocument = list.RowTemplate
Dim listRowTable As Table = listRow.Tables(0)
Dim listRowCells As TableCellCollection = listRowTable.FirstRow.Cells
listRow.CreateSnText(listRowCells.InsertAfter(2).ContentRange.End, "Quanti
list.EndUpdate()
list.Field.Update()
```

How to: Format Data

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Format Data](#)

The following example demonstrates how to format data using the Snap Report API.

- To start formatting document content, you need to prepare the document for modification by doing the following.
1. Load the document template using the RichEditDocumentServer.LoadDocumentTemplate method.
 2. Call the ISnapFieldOwner.CreateSnList method to create a new Snap list or retrieve one of the existing lists by calling the ISnapFieldOwner.FindListByName method.
 3. Convert the document fields to Snap Fields. To do that, call the ISnapFieldOwner.ParseField method. Note that without parsing, you won't be able to modify the document.
 4. Enable the modifying of the list by calling the SnapEntity.BeginUpdate method.

Every part of a snap document, whether it's a list header or a row template, is represented by a table of document fields. Tables of the list header can be retrieved through the SnapList.ListHeader property, and the tables in the row template - through the SnapList.RowTemplate property.


To apply text formatting to cell content, call the SubDocument.BeginUpdateCharacters method for the specified range, use the returned CharacterProperties instance to modify the required properties and call the SubDocument.EndUpdateCharacters method to finalize the modification.

You can change table appearance by changing the color of different table elements. To achieve the required result, the following properties can be used.

- TableCell.BackgroundColor - sets the cell background color
- TableCellBorder.LineColor - sets the cell border color
- Table.TableBackgroundColor - sets the color of the empty space between cells

You can also apply one of the predefined themes to the whole document. To do that, call the SnapDocument.ApplyTheme method. Pass one of the Themes enumeration values as a method parameter.

Call the SnapEntity.EndUpdate method to finalize the list modification. Then, update the list fields by calling the FieldCollection.Update method.

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataShapingActions.cs)
server.LoadDocumentTemplate("Template.snx");
SnapList list = server.Document.FindListByName("Data Source 11");
server.Document.ParseField(list.Field);
list.BeginUpdate();
list.EditorRowLimit = 100500;
SnapDocument header = list.ListHeader;
Table headerTable = header.Tables[0];
headerTable.SetPreferredWidth(50 * 100, WidthType.FiftiethsOfPercent);
foreach (TableRow row in headerTable.Rows)
{
    foreach (TableCell cell in row.Cells)
    {
        // Apply cell formatting.
        cell.Borders.Left.LineColor = System.Drawing.Color.White;
        cell.Borders.Right.LineColor = System.Drawing.Color.White;
        cell.Borders.Top.LineColor = System.Drawing.Color.White;
        cell.Borders.Bottom.LineColor = System.Drawing.Color.White;
        cell.BackgroundColor = System.Drawing.Color.SteelBlue;
        // Apply text formatting.
        CharacterProperties formatting = header.BeginUpdateCharacters(cell.ContentRange);
        formatting.Bold = true;
        formatting.ForeColor = System.Drawing.Color.White;
        header.EndUpdateCharacters(formatting);
    }
}
// Customize the row template.
SnapDocument rowTemplate = list.RowTemplate;
Table rowTable = rowTemplate.Tables[0];
rowTable.SetPreferredWidth(50 * 100, WidthType.FiftiethsOfPercent);
foreach (TableRow row in rowTable.Rows)
{
    foreach (TableCell cell in row.Cells)
    {
        cell.Borders.Left.LineColor = System.Drawing.Color.Transparent;
        cell.Borders.Right.LineColor = System.Drawing.Color.Transparent;
        cell.Borders.Top.LineColor = System.Drawing.Color.Transparent;
        cell.Borders.Bottom.LineColor = System.Drawing.Color.LightGray;
    }
}
list.EndUpdate();
list.Field.Update();
```

Visual Basic

```
(DataShapingActions.vb)
server.LoadDocumentTemplate("Template.snx")
Dim list As SnapList = server.Document.FindListByName("Data Source 11")
server.Document.ParseField(list.Field)
list.BeginUpdate()
list.EditorRowLimit = 100500
Dim header As SnapDocument = list.ListHeader
Dim headerTable As Table = header.Tables(0)
headerTable.SetPreferredWidth(50 * 100, WidthType.FiftiethsOfPercent)
For Each row As TableRow In headerTable.Rows
    For Each cell As TableCell In row.Cells
        ' Apply cell formatting.
        cell.Borders.Left.LineColor = System.Drawing.Color.White
        cell.Borders.Right.LineColor = System.Drawing.Color.White
        cell.Borders.Top.LineColor = System.Drawing.Color.White
        cell.Borders.Bottom.LineColor = System.Drawing.Color.White
        cell.BackgroundColor = System.Drawing.Color.SteelBlue
        ' Apply text formatting.
        Dim formatting As CharacterProperties = header.BeginUpdateCharacter
        formatting.Bold = True
        formatting.ForeColor = System.Drawing.Color.White
        header.EndUpdateCharacters(formatting)
    Next cell
Next row
' Customize the row template.
Dim rowTemplate As SnapDocument = list.RowTemplate
Dim rowTable As Table = rowTemplate.Tables(0)
rowTable.SetPreferredWidth(50 * 100, WidthType.FiftiethsOfPercent)
For Each row As TableRow In rowTable.Rows
    For Each cell As TableCell In row.Cells
        cell.Borders.Left.LineColor = System.Drawing.Color.Transparent
        cell.Borders.Right.LineColor = System.Drawing.Color.Transparent
        cell.Borders.Top.LineColor = System.Drawing.Color.Transparent
        cell.Borders.Bottom.LineColor = System.Drawing.Color.LightGray
    Next cell
Next row
list.EndUpdate()
list.Field.Update()
```

The following image illustrates the result of document formatting.

ProductName	UnitPrice	UnitsInStock	Discontinued
Chai	\$18.00	39	<input type="checkbox"/>
Chang			
Aniseed Syrup			
Chef Anton's Cajun Seasoning			
Chef Anton's Gumbo Mix			
Grandma's Boysenberry Spread			

ProductName	UnitPrice	UnitsInStock	Discontinued
Chai	\$18.00	39	<input type="checkbox"/>
Chang	\$19.00	17	<input type="checkbox"/>
Aniseed Syrup	\$10.00	13	<input type="checkbox"/>
Chef Anton's Cajun Seasoning	\$22.00	53	<input type="checkbox"/>
Chef Anton's Gumbo Mix	\$21.35	0	<input checked="" type="checkbox"/>
Grandma's Boysenberry Spread	\$25.00	120	<input type="checkbox"/>

How to: Group Data

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Group Data](#)

The following example demonstrates how to group Snap document data using the Snap Report API.

To start formatting document content, prepare the document for modification by doing the following.

1. Load the document template using the RichEditDocumentServer.LoadDocumentTemplate method.
2. Call the ISnapFieldOwner.CreateSnList method to create a new Snap list or retrieve one of the existed lists by calling the ISnapFieldOwner.FindListByName method.
3. Convert the document fields to Snap Fields. To do that, call the ISnapFieldOwner.ParseField method. Note that without parsing, you won't be able to modify the document.
4. Start modifying the list by calling the SnapEntity.BeginUpdate method.

The next step is to start grouping data. To do that, perform the following steps.

1. Create a new SnapListGroupParam object. The name of the field to be used as a grouping criteria and a sort order type must be passed to the object's constructor method.
2. Create a new SnapListGroupInfo object by calling the SnapListGroups.CreateSnapListGroupInfo method with the passed SnapListGroupParam object.
3. Add the created SnapListGroupInfo object to the corresponding list collection. To do that, access the collection through the SnapList.Groups property and call the **Add** method.

You can also create a header and footer for the group. To complete this task, follow the steps below.

- To create a group header/footer, call the SnapListGroupInfo.CreateHeader or SnapListGroupInfo.CreateFooter method.
- Then, create a table in the header/footer. To do that, add a new item to the header/footer collection of tables, accessible though the SubDocument.Tables property, by calling the **Create** method.
- To insert the field to the table cell, call the ISnapFieldOwner.CreateSnText method. Use the position within the desired cell as a method parameter. The cell range is accessible through the TableCell.ContentRange property.
- You can customize the table by changing its border and background color. To do that, use the TableCellBorder.LineColor and TableCell.BackgroundColor properties.

To finalize the modification, call the SnapEntity.EndUpdate method. Call the FieldCollection.Update method to update the list fields. Note that without updating, the created modifications will not take effect.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataShapingActions.cs)
server.LoadDocumentTemplate("Template.snx");
SnapList list = server.Document.FindListByName("Data Source 11");
server.Document.ParseField(list.Field);
list.BeginUpdate();
list.EditorRowLimit = 100500;
// Add a header to the Snap list.
SnapDocument listHeader = list.ListHeader;
Table listHeaderTable = listHeader.Tables[0];
SnapListGroupInfo group = list.Groups.CreateSnapListGroupInfo(
    new SnapListGroupParam("CategoryID", ColumnSortOrder.Ascending));
list.Groups.Add(group);
// Add a group header.
SnapDocument groupHeader = group.CreateHeader();
Table headerTable = groupHeader.Tables.Create(groupHeader.Range.End, 1, 1);
headerTable.SetPreferredWidth(50 * 100, WidthType.FiftiethsOfPercent);
TableCellCollection groupHeaderCells = headerTable.FirstRow.Cells;
groupHeader.InsertText(groupHeaderCells[0].ContentRange.End, "Category ID: ");
groupHeader.CreateSnText(groupHeaderCells[0].ContentRange.End, "CategoryID");
// Customize the group header formatting.
groupHeaderCells[0].BackgroundColor = System.Drawing.Color.LightGray;
groupHeaderCells[0].Borders.Bottom.LineColor = System.Drawing.Color.White;
groupHeaderCells[0].Borders.Left.LineColor = System.Drawing.Color.White;
groupHeaderCells[0].Borders.Right.LineColor = System.Drawing.Color.White;
groupHeaderCells[0].Borders.Top.LineColor = System.Drawing.Color.White;
// Add a group footer.
SnapDocument groupFooter = group.CreateFooter();
Table footerTable = groupFooter.Tables.Create(groupFooter.Range.End, 1, 1);
footerTable.SetPreferredWidth(50 * 100, WidthType.FiftiethsOfPercent);
TableCellCollection groupFooterCells = footerTable.FirstRow.Cells;
groupFooter.InsertText(groupFooterCells[0].ContentRange.End, "Count = ");
groupFooter.CreateSnText(groupFooterCells[0].ContentRange.End,
    @"CategoryID \sr Group \sf Count");
// Customize the group footer formatting.
groupFooterCells[0].BackgroundColor = System.Drawing.Color.LightGray;
groupFooterCells[0].Borders.Bottom.LineColor = System.Drawing.Color.White;
groupFooterCells[0].Borders.Left.LineColor = System.Drawing.Color.White;
groupFooterCells[0].Borders.Right.LineColor = System.Drawing.Color.White;
groupFooterCells[0].Borders.Top.LineColor = System.Drawing.Color.White;
list.EndUpdate();
//list.Field.Update();
```

Visual Basic

```
(DataShapingActions.vb)
server.LoadDocumentTemplate("Template.snx")
Dim list As SnapList = server.Document.FindListByName("Data Source 11")
server.Document.ParseField(list.Field)
list.BeginUpdate()
list.EditorRowLimit = 100500
' Add a header to the Snap list.
Dim listHeader As SnapDocument = list.ListHeader
Dim listHeaderTable As Table = listHeader.Tables(0)
Dim group As SnapListGroupInfo = list.Groups.CreateSnapListGroupInfo(New S
list.Groups.Add(group)
' Add a group header.
Dim groupHeader As SnapDocument = group.CreateHeader()
Dim headerTable As Table = groupHeader.Tables.Create(groupHeader.Range.End
headerTable.SetPreferredWidth(50 * 100, WidthType.FiftiethsOfPercent)
Dim groupHeaderCells As TableCellCollection = headerTable.FirstRow.Cells
groupHeader.InsertText(groupHeaderCells(0).ContentRange.End, "Category ID:
groupHeader.CreateSnText(groupHeaderCells(0).ContentRange.End, "CategoryID
' Customize the group header formatting.
groupHeaderCells(0).BackColor = System.Drawing.Color.LightGray
groupHeaderCells(0).Borders.Bottom.LineColor = System.Drawing.Color.White
groupHeaderCells(0).Borders.Left.LineColor = System.Drawing.Color.White
groupHeaderCells(0).Borders.Right.LineColor = System.Drawing.Color.White
groupHeaderCells(0).Borders.Top.LineColor = System.Drawing.Color.White
' Add a group footer.
Dim groupFooter As SnapDocument = group.CreateFooter()
Dim footerTable As Table = groupFooter.Tables.Create(groupFooter.Range.End
footerTable.SetPreferredWidth(50 * 100, WidthType.FiftiethsOfPercent)
Dim groupFooterCells As TableCellCollection = footerTable.FirstRow.Cells
groupFooter.InsertText(groupFooterCells(0).ContentRange.End, "Count = ")
groupFooter.CreateSnText(groupFooterCells(0).ContentRange.End, "CategoryID
' Customize the group footer formatting.
groupFooterCells(0).BackColor = System.Drawing.Color.LightGray
groupFooterCells(0).Borders.Bottom.LineColor = System.Drawing.Color.White
groupFooterCells(0).Borders.Left.LineColor = System.Drawing.Color.White
groupFooterCells(0).Borders.Right.LineColor = System.Drawing.Color.White
groupFooterCells(0).Borders.Top.LineColor = System.Drawing.Color.White
list.EndUpdate()
list.Field.Update();
```

How to: Filter Data

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Filter Data](#)

The following example describes how to filter data within a Snap document using the following steps.

1. Load the document template using the RichEditDocumentServer.LoadDocumentTemplate method.
2. Call the ISnapFieldOwner.CreateSnList method to create a new Snap list or retrieve one of the existing lists by calling the ISnapFieldOwner.FindListByName method.
3. Convert the document fields to Snap Fields. To do that, call the ISnapFieldOwner.ParseField method. Note that with this step skipped, you won't be able to modify the document.
4. Start modifying the list by calling the SnapEntity.BeginUpdate method.
5. To apply filtering to the list, add the required string filter expression to the list collection of filter settings, accessible through SnapList.Filters property.
6. Call the SnapEntity.EndUpdate method to finalize the list modification and update the list fields by calling the FieldCollection.Update method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataShapingActions.cs)
// Delete the document's content.
server.LoadDocument("Template.snx");
SnapList list = server.Document.FindListByName("Data Source 11");
server.Document.ParseField(list.Field);
list.BeginUpdate();
// Apply filtering:
const string filter = "[Discontinued] = False";
if (!list.Filters.Contains(filter))
{
    list.Filters.Add(filter);
}
list.EndUpdate();
list.Field.Update();
```

Visual Basic

```
(DataShapingActions.vb)
' Delete the document's content.
server.LoadDocument("Template.snx")
Dim list As SnapList = server.Document.FindListByName("Data Source 11")
server.Document.ParseField(list.Field)
list.BeginUpdate()
' Apply filtering:
Const filter As String = "[Discontinued] = False"
If Not list.Filters.Contains(filter) Then
    list.Filters.Add(filter)
End If
list.EndUpdate()
list.Field.Update()
```

How to: Sort Data

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Sort Data](#)

The following example demonstrates how to sort Snap document data using the following steps.

1. Load the document template using the RichEditDocumentServer.LoadDocumentTemplate method.
2. Call the ISnapFieldOwner.CreateSnList method to create a new Snap list or retrieve one of the existing lists by calling the ISnapFieldOwner.FindListByName method.
3. Convert the document fields to Snap Fields. To do that, call the ISnapFieldOwner.ParseField method. Note that without performing this step, you won't be able to modify the document.
4. Enable modifying the list by calling the SnapEntity.BeginUpdate method.
5. Create a new sorting parameter represented by the SnapListGroupParam instance. Pass the target field name and the desired sort order to the SnapListGroupParam object's constructor method.
6. Add the created object to the corresponding list collection. To do that, access the collection through the SnapList.Sorting property and call the **Add** method.
7. Call the SnapEntity.EndUpdate method to finalize the list modification. Then, update the list fields by calling the FieldCollection.Update method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

(DataShapingActions.cs)
server.LoadDocumentTemplate("Template.snx");
SnapList list = server.Document.FindListByName("Data Source 11");
server.Document.ParseField(list.Field);
list.BeginUpdate();
list.Sorting.Add(new SnapListGroupParam("UnitPrice", ColumnSortOrder.Descending));
list.EndUpdate();
list.Field.Update();

Visual Basic

(DataShapingActions.vb)
server.LoadDocumentTemplate("Template.snx")
Dim list As SnapList = server.Document.FindListByName("Data Source 11")
server.Document.ParseField(list.Field)
list.BeginUpdate()
list.Sorting.Add(New SnapListGroupParam("UnitPrice", ColumnSortOrder.Desce
list.EndUpdate()
list.Field.Update()

How to: Create Parameter

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Create Parameter](#)

The following example demonstrates how to create the report parameter in the Snap Report API.

- 1. Call the `IRichEditDocumentServer.LoadDocumentTemplate` method to load the document template and open it for modification by calling the `SubDocument.BeginUpdate` method.
- 2. Create a new `Parameter` instance that is the new report parameter.
- 3. To set the parameter's name, type and value, use the `Parameter.Name`, `Parameter.Type` and `Parameter.Value` properties.
- 4. Add the created object to the document's parameters collection, accessible through the `SnapDocument.Parameters` property.
- 5. Finalize the modification by calling the `SubDocument.EndUpdate` method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataVisualizationToolsActions.cs)
server.LoadDocumentTemplate("Template.snx");
server.Document.BeginUpdate();
//Create parameter:
Parameter param1 = new Parameter();
param1.Name = "Region";
param1.Type = typeof(System.String);
param1.Value = "NEW ENGLAND";
server.Document.Parameters.Add(param1);
//Insert parameter field in the document:
SnapList list = server.Document.FindListByName("Data Source 11");
server.Document.ParseField(list.Field);
list.BeginUpdate();
list.ListHeader.InsertText(list.ListHeader.Tables[0].FirstRow.Cells.InsertAfter(3).ContentRange.End, "Reg:");
list.RowTemplate.CreateSnText(list.RowTemplate.Tables[0].FirstRow.Cells.InsertAfter(3).ContentRange.End, (
list.Field.Update();
list.EndUpdate();
server.Document.EndUpdate();
server.Document.Fields.Update();
```

Visual Basic

```
(DataVisualizationToolsActions.vb)
server.LoadDocumentTemplate("Template.snx")
server.Document.BeginUpdate()
'Create parameter:
Dim param1 As New Parameter()
param1.Name = "Region"
param1.Type = GetType(System.String)
param1.Value = "NEW ENGLAND"
server.Document.Parameters.Add(param1)
'Insert parameter field in the document:
Dim list As SnapList = server.Document.FindListByName("Data Source 11")
server.Document.ParseField(list.Field)
list.BeginUpdate()
list.ListHeader.InsertText(list.ListHeader.Tables(0).FirstRow.Cells.Insert
list.RowTemplate.CreateSnText(list.RowTemplate.Tables(0).FirstRow.Cells.In
list.Field.Update()
list.EndUpdate()
server.Document.EndUpdate()
server.Document.Fields.Update()
```

How to: Create Calculated Field

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Create Calculated Field](#)

This example demonstrates how to create a calculated field in the Snap Report API.

- 1. Load the document template by calling the `IRichEditDocumentServer.LoadDocumentTemplate` method.
- 2. Call the `SubDocument.BeginUpdate` method to enable the document modification.
- 3. Create a new `CalculatedField` object. Pass the name for the created field and data member to be associated with to the object's constructor method.
- 4. To associate the calculated field with the data source, use the `CalculatedField.DataSourceName` property.
- 5. Specify the type and expression for the calculated field by setting the `CalculatedField.FieldType` and `CalculatedField.Expression` properties.
- 6. Add the created `CalculatedField` object to the document collection of calculated fields. To do that, access the collection through the `IDataSourceOwner.CalculatedFields` property and call the **Add** method.
- 7. To finalize the modification, call the `SubDocument.EndUpdate` method.
- 8. Update the document fields by calling the `FieldCollection.Update` method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataVisualizationToolsActions.cs)
server.LoadDocumentTemplate("Template.snx");
server.Document.BeginUpdate();
CalculatedField field = new CalculatedField("newField", "Products");
field.FieldType = DevExpress.XtraReports.UI.FieldType.Int32;
field.Expression = "[UnitsInStock]*[UnitPrice]";
field.DataSourceName = "Data Source 1";
server.Document.CalculatedFields.Add(field);
server.Document.EndUpdate();
server.Document.Fields.Update();
```

Visual Basic

```
(DataVisualizationToolsActions.vb)
server.LoadDocumentTemplate("Template.snx")
server.Document.BeginUpdate()
Dim field As New CalculatedField("newField", "Products")
field.FieldType = DevExpress.XtraReports.UI.FieldType.Int32
field.Expression = "[UnitsInStock]*[UnitPrice]"
field.DataSourceName = "Data Source 1"
server.Document.CalculatedFields.Add(field)
server.Document.EndUpdate()
server.Document.Fields.Update()
```

How to: Create Barcode

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Create Barcode](#)

To create a barcode in a snap document, do the following:

- 1. Call the `IRichEditDocumentServer.LoadDocumentTemplate` method to load the document template and open it for modification by calling the `SubDocument.BeginUpdateCharacters` method.
- 2. Create a new `SnapBarCode` object by calling the `ISnapFieldOwner.CreateSnBarCode` method with the barcode placement position passed as a parameter.
- 3. Call the `ISnapFieldOwner.ParseField` to convert the barcode field to a Snap Field. Note that if this step is skipped, you won't be able to modify the barcode.
- 4. Call the `SnapEntity.BeginUpdate` method to open the barcode for modification.
- 5. Specify the desired barcode settings, such as orientation (`SnapBarCode.Orientation`), module (`SnapBarCode.Module`), and alignment (`SnapBarCode.Alignment`), etc.
- 6. Finalize the modification by calling the `SnapEntity.EndUpdate` method.
- 7. Call the `FieldCollection.Update` method to update the document fields.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataVisualizationToolsActions.cs)
server.LoadDocumentTemplate("Template.snx");
server.Document.BeginUpdate();
SnapBarCode barcode = server.Document.CreateSnBarCode(server.Document.Range.Start);
server.Document.ParseField(barcode.Field);
barcode.BeginUpdate();
barcode.DataFieldName = "EAN13";
barcode.Module = 10;
barcode.Orientation = DevExpress.XtraPrinting.BarCode.BarCodeOrientation.Normal;
barcode.AutoModule = true;
barcode.ShowData = false;
barcode.EndUpdate();
barcode.Field.Update();
```

Visual Basic

```
(DataVisualizationToolsActions.vb)
server.LoadDocumentTemplate("Template.snx")
server.Document.BeginUpdate()
Dim barcode As SnapBarCode = server.Document.CreateSnBarCode(server.Docume
server.Document.ParseField(barcode.Field)
barcode.BeginUpdate()
barcode.DataFieldName = "EAN13"
barcode.Module = 10
barcode.Orientation = DevExpress.XtraPrinting.BarCode.BarCodeOrientation.N
barcode.AutoModule = True
barcode.ShowData = False
barcode.EndUpdate()
barcode.Field.Update()
```

How to: Create Sparkline

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Create Sparkline](#)

The following example demonstrates how to create a sparkline in the Snap Report API.

1. Call the `IRichEditDocumentServer.LoadDocumentTemplate` method to load the document template and open it for modification by calling the `SubDocument.BeginUpdateCharacters` method.
2. Create a new `SnapSparkline` object by calling the `ISnapFieldOwner.CreateSnSparkline` method with the passed position to place the sparkline.
3. Convert the created sparkline field to a `Snap Field`. To do that, use the `ISnapFieldOwner.ParseField` method. Note that without conversion, you will not be able to modify the sparkline.
4. Start modifying the sparkline by calling the `SnapEntity.BeginUpdate` method.
5. Specify the desired sparkline settings, such as view type (`SnapSparkline.ViewType`), color (`SnapSparkline.Color`), dimensions (`SnapSparkline.Size`), etc.
6. Finalize the modification using the `SnapEntity.EndUpdate` method.
7. Call the `FieldCollection.Update` method to update the document fields.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataVisualizationToolsActions.cs)
server.LoadDocumentTemplate("Template.snx");
server.Document.BeginUpdate();
SnapSparkline sparkline = server.Document.CreateSnSparkline(server.Document.Range.End, "UnitsInStock");
server.Document.ParseField(sparkline.Field);
sparkline.BeginUpdate();
sparkline.DataSourceName = "/Data Source 1.Products";
sparkline.ViewType = SparklineViewType.Line;
sparkline.Color = System.Drawing.Color.Teal;
sparkline.EndUpdate();
sparkline.Field.Update();
```

Visual Basic

```
(DataVisualizationToolsActions.vb)
server.LoadDocumentTemplate("Template.snx")
server.Document.BeginUpdate()
Dim sparkline As SnapSparkline = server.Document.CreateSnSparkline(server.Document.Range.End, "UnitsInStock")
server.Document.ParseField(sparkline.Field)
sparkline.BeginUpdate()
sparkline.DataSourceName = "/Data Source 1.Products"
sparkline.ViewType = SparklineViewType.Line
sparkline.Color = System.Drawing.Color.Teal
sparkline.EndUpdate()
sparkline.Field.Update()
```

See Also
Sparkline

How to: Create Check Box

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Create Check Box](#)

The following example demonstrates how to create a check box in the Snap document.

1. Call the `IRichEditDocumentServer.LoadDocumentTemplate` method to load the document template to the server.
2. Enable the document modification by calling the `SubDocument.BeginUpdate` method.
3. Create new `SnapCheckBox` instance by calling the `ISnapFieldOwner.CreateSnCheckBox` method. Pass the position to place the check box and the data field to which it corresponds to the method.
4. Convert the check box field to a Snap field. To do that, call the `ISnapFieldOwner.ParseField` method. Note that if this step is skipped, you won't be able to modify the check box.
5. Start the checkbox modification by calling the `SnapEntity.BeginUpdate` method.
6. Specify the check state by setting the `SnapCheckBox.State` property.
7. To finalize the modification, call `SnapEntity.EndUpdate` method.
8. Call the `FieldCollection.Update` method to update the check box field.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T429162>.

C#

```
(DataVisualizationToolsActions.cs)
server.LoadDocumentTemplate("Template.snx");
server.Document.BeginUpdate();
SnapCheckBox checkbox = server.Document.CreateSnCheckBox(server.Document.Range.Start, "Discontinued");
server.Document.ParseField(checkbox.Field);
checkbox.BeginUpdate();
checkbox.State = System.Windows.Forms.CheckState.Checked;
checkbox.EndUpdate();
checkbox.Field.Update();
```

How to: Generate Master-Detail Mail Merge Documents

[Office File API](#) > [Snap Report API](#) > [Examples](#) > [How to: Generate Master-Detail Mail Merge Documents](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E5078>.

The following code uses the Snap Report API to generate a mail-merge report in the context of the [DevExpress Office File API](#).

It is a console application that creates a document and then exports it to an RTF file.

To use the [SnapDocumentServer](#), the project requires references to the following assemblies:

- DevExpress.Data.v18.1.dll
- DevExpress.Docs.v18.1.dll
- DevExpress.Office.v18.1.Core.dll
- DevExpress.RichEdit.v18.1.Core.dll
- DevExpress.Snap.v18.1.Core.dll

To perform mail merge, the application requires a template - a file in the native .SNX format which contains the document layout, specific fields and data source information. The template file must be created beforehand using the Snap Design Surface. For an example of template creation, review the Lesson 3 - Create a Mail Merge Report.

The code handles the [SnapDocumentServer.SnapMailMergeRecordStarted](#) and [SnapDocumentServer.SnapMailMergeRecordFinished](#) events to make adjustments to the merged document created for the particular data record.

The server loads a mail merge template using the RichEditDocumentServer.LoadDocument method. Subsequently, the code assigns the data source by specifying the IDataSourceOwner.DataSource property. The mail merge operation is performed by calling the [SnapDocumentServer.SnapMailMerge](#) method.

C#

```
(Program.cs)
using DevExpress.Snap;
using DevExpress.Snap.Core.API;
using DevExpress.XtraRichEdit;
using MailMergeServer.nwindDataSetTableAdapters;
using System;
using System.Data.OleDb;
using System.IO;
// ...

SnapDocumentServer server = new SnapDocumentServer();
server.SnapMailMergeRecordStarted += server_SnapMailMergeRecordStarted;
server.SnapMailMergeRecordFinished += server_SnapMailMergeRecordFinished;
server.LoadDocument(templateFileName);
object dataSource = CreateDataSource();
server.Document.DataSource = dataSource;
Console.WriteLine("Performing mail merge... ");
server.SnapMailMerge(outputFileName, DocumentFormat.Rtf);
```

Zip Compression and Archive API

[Office File API](#) > [Zip Compression and Archive API](#)

A Zip Compression and Archive API is designed for data compression and archive generation. It implements **Deflate** data compression algorithm and creates **zip** archives compatible with PKWARE specification 2.0. In addition, the DevExpress Zip Compression and Archive API supports AES encryption up to 256 bit. The maximum size of uncompressed data for a single file is 4 GB (32-bit number maximum value).

Important

The Universal Subscription or an additional Office File API Subscription is required to use this component or library in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Main Features

- Create new zip files or update existing zip files on disk or in memory
- Zip or unzip to and from disks or memory
- Compress and decompress .NET streams and byte arrays.
- Zip file password encryption with support for AES (128, 192, and 256 bit encryption)
- Allow different encryption passwords for each file
- Set individual file comments
- Allow file overwrite on a per file basis
- Programmatically filter files to process
- Progress tracking mechanism that allows you to cancel archive operations
- Comprehensive API that enables you to control each archive item and all actions
- Object model designed for easy extensibility

Examples

- [Getting Started](#)
- [Examples](#)

Getting Started

[Office File API](#) > [Zip Compression and Archive API](#) > [Getting Started](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

To get started with a Zip Compression and Archive API, perform the following steps.

1. Create a new Windows Forms Application project.
2. Add reference to the **DevExpress.Docs.v18.1.dll** library.
3. Create a method containing the code listed below.

C#

```
using DevExpress.Compression;
// ...
string zipFileName = "Test.zip";
string sourceDir = @"C:\Users\Public\Documents\DevExpress Demos 18.1\Components\Data";
string password = "123";
EncryptionType encryptionType = EncryptionType.PkZip;
using (ZipArchive archive = new ZipArchive())
{
    archive.Password = password;
    archive.EncryptionType = encryptionType;
    archive.AddDirectory(sourceDir);
    archive.Save(zipFileName);
}
```

Visual Basic

```
Imports DevExpress.Compression
' ...
Private zipFileName As String = "Test.zip"
Private sourceDir As String = "C:\Users\Public\Documents\DevExpress Demos 18.1\Components\Data"
Private password As String = "123"
Private encryptionType As EncryptionType = EncryptionType.PkZip
Using archive As New ZipArchive()
    archive.Password = password
    archive.EncryptionType = encryptionType
    archive.AddDirectory(sourceDir)
    archive.Save(zipFileName)
End Using
```

4. Run the project and execute the method. It creates the Test.zip archive with all files encrypted using "123" password.

See Also

[Training Videos](#)
[Examples](#)

Examples

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#)

This section provides a list of examples contained in this help.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

- [How to: Archive a Directory](#)
- [How to: Archive Selected Files](#)
- [How to: Add File to File Archive](#)
- [How to: Filter Files to Archive](#)
- [How to: Cancel Archiving](#)
- [How to: Password Protect Archive Files](#)
- [How to: Add Comment to File](#)
- [How to: Compress .NET Stream](#)
- [How to: Compress Byte Array](#)
- [How to: Unzip an Archive](#)
- [How to: Resolve File Conflicts when Unzipping](#)

How to: Archive a Directory

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Archive a Directory](#)

To archive a directory, do the following:

1. Create a [ZipArchive](#) class instance.
2. Use its [ZipArchive.AddDirectory](#) method to specify the source directory.
3. Call the [ZipArchive.Save](#) method to create an archive and save it to a specified location.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveDirectory() {
    string path = this.startupPath;
    using (ZipArchive archive = new ZipArchive()) {
        archive.AddDirectory(path);
        archive.Save("Documents\\ArchiveDirectory.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveDirectory()
    Dim path As String = Me.startupPath
    Using archive As New ZipArchive()
        archive.AddDirectory(path)
        archive.Save("Documents\\ArchiveDirectory.zip")
    End Using
End Sub
```

How to: Archive Selected Files

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Archive Selected Files](#)

To archive selected files, do the following:

1. Create a [ZipArchive](#) class instance.
2. Call its [ZipArchive.AddFile](#) method for each selected file.
3. To store a file without its path, call the [ZipArchive.AddFile](#) method with the "/" parameter. The file will be placed in the root node of the archive.
4. Call the [ZipArchive.Save](#) method. The archive will be created and saved to a specified location.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This code snippet demonstrates how to create a new archive and add files to the archive root. The [ZipArchive.Save](#) method compresses data and saves the archive to a file.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveFiles() {
    string[] sourcefiles = this.sourceFiles;
    using (ZipArchive archive = new ZipArchive()) {
        foreach (string file in sourcefiles) {
            archive.AddFile(file, "/");
        }
        archive.Save("Documents\\ArchiveFiles.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveFiles()
    Dim sourcefiles() As String = Me.sourceFiles
    Using archive As New ZipArchive()
        For Each file As String In sourcefiles
            archive.AddFile(file, "/")
        Next file
        archive.Save("Documents\\ArchiveFiles.zip")
    End Using
End Sub
```

How to: Add File to File Archive

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Add File to File Archive](#)

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

When the file archive is opened with the [ZipArchive.Read](#) method, you cannot save it to the same file. The [ZipArchive.Save](#) method attempts to overwrite the locked file resulting in an exception.

This code snippet illustrates how you can add a file to an archive and save it with the same name as before.

C#

```
(ZipExamples.cs)
public void AddFileToArchive() {
    MemoryStream stream = new MemoryStream();
    string[] sourcefiles = this.sourceFiles;
    string pathToZipArchive = "Documents\\Example.zip";
    using (FileStream fs = File.Open(pathToZipArchive, FileMode.Open)) {
        fs.CopyTo(stream);
        fs.Close();
    }
    stream.Seek(0, SeekOrigin.Begin);
    using (ZipArchive archive = ZipArchive.Read(stream, System.Text.Encoding.Default, false)) {
        foreach (string sfile in sourcefiles) {
            archive.AddFile(sfile, "/");
        }
        archive.Save(pathToZipArchive);
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Public Sub AddFileToArchive()
    Dim stream As New MemoryStream()
    Dim sourcefiles() As String = Me.sourceFiles
    Dim pathToZipArchive As String = "Documents\\Example.zip"
    Using fs As FileStream = File.Open(pathToZipArchive, FileMode.Open)
        fs.CopyTo(stream)
        fs.Close()
    End Using
    stream.Seek(0, SeekOrigin.Begin)
    Using archive As ZipArchive = ZipArchive.Read(stream, System.Text.Encoding.Default, false)
        For Each sfile As String In sourcefiles
            archive.AddFile(sfile, "/")
        Next sfile
        archive.Save(pathToZipArchive)
    End Using
End Sub
```

How to: Filter Files to Archive

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Filter Files to Archive](#)

To only archive files that meet a certain criteria, do the following:

1. Create a [ZipArchive](#) class instance.
2. Subscribe to its [ZipArchive.ItemAdding](#) event. This event is raised for each [ZipItem](#) that holds information for a file or directory. Add event handler code to analyze the ZipItem and specify the action to perform to continue archiving, to stop the process or to cancel adding this particular item to an archive.
3. Call the [ZipArchive.AddFile](#) method for each selected file.
4. Call the [ZipArchive.Save](#) method to create an archive and save it to a specified location.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example demonstrates how to handle the [ZipArchive.ItemAdding](#) event to decide for each file whether it should be included in the archive.

If a file creation date is not the current date, the file is not added to the archive. A volatile variable is used to indicate whether the process should be stopped - it can be useful to interrupt archive creation if too many files are specified.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;
    volatile bool stopArchiving = false;
    public void FilterArchiveFiles() {
        string[] sourcefiles = this.sourceFiles;
        using (ZipArchive archive = new ZipArchive()) {
            archive.ItemAdding += archive_ItemAdding;
            foreach (string file in sourceFiles) {
                archive.AddFile(file, "/");
            }
            archive.Save("Documents\\FilterArchiveFiles.zip");
        }
    }
    private void archive_ItemAdding(object sender, ZipItemAddingEventArgs args) {
        if (args.Item.CreationTime.Date != DateTime.Today)
            args.Action = ZipItemAddingAction.Cancel;
        if (stopArchiving) args.Action = ZipItemAddingAction.Stop;
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression
'INSTANT VB TODO TASK: There is no VB.NET equivalent to 'volatile':
'ORIGINAL LINE: volatile bool stopArchiving = False;
Private stopArchiving As Boolean = False
Public Sub FilterArchiveFiles()
    Dim sourcefiles() As String = Me.sourceFiles
    Using archive As New ZipArchive()
        AddHandler archive.ItemAdding, AddressOf archive_ItemAdding
        For Each file As String In Me.sourceFiles
            archive.AddFile(file, "/" & file)
        Next file
        archive.Save("Documents\FolderArchiveFiles.zip")
    End Using
End Sub
Private Sub archive_ItemAdding(ByVal sender As Object, ByVal args As ZipItemAddingEventArgs)
    If args.Item.CreationTime.Date <> DateTime.Today Then
        args.Action = ZipItemAddingAction.Cancel
    End If
    If stopArchiving Then
        args.Action = ZipItemAddingAction.Stop
    End If
End Sub
End Sub
```

How to: Cancel Archiving

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Cancel Archiving](#)

To provide the capability to cancel an archiving process after it has been started, do the following:

1. Create a [ZipArchive](#) class instance.
2. Subscribe to the [ZipArchive.Progress](#) event.
3. Set the [CanContinueEventArgs.CanContinue](#) property of event arguments to **false**, in case you decide to stop the operation.
4. Call the [ZipArchive.AddFile](#) method for each selected file.
5. Call the [ZipArchive.Save](#) method to create an archive and save it to a specified location.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

volatile bool stopProgress = false;
public void CancelArchiveProgress() {
    string[] sourcefiles = this.sourceFiles;
    using (ZipArchive archive = new ZipArchive()) {
        archive.Progress += archive_Progress;
        foreach (string file in sourceFiles) {
            archive.AddFile(file, "/");
        }
        archive.Save("Documents\\CancelArchiveProgress.zip");
    }
}

private void archive_Progress(object sender, ProgressEventArgs args) {
    args.CanContinue = !this.stopProgress;
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

'INSTANT VB TODO TASK: There is no VB.NET equivalent to 'volatile':
'ORIGINAL LINE: volatile bool stopProgress = False;
Private stopProgress As Boolean = False
Public Sub CancelArchiveProgress()
    Dim sourcefiles() As String = Me.sourceFiles
    Using archive As New ZipArchive()
        AddHandler archive.Progress, AddressOf archive_Progress
        For Each file As String In Me.sourceFiles
            archive.AddFile(file, "/")
        Next file
        archive.Save("Documents\\CancelArchiveProgress.zip")
    End Using
End Sub

Private Sub archive_Progress(ByVal sender As Object, ByVal args As
    args.CanContinue = Not Me.stopProgress
End Sub
```

How to: Password Protect Archive Files

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Password Protect Archive Files](#)

To encrypt and password protect files in an archive, do the following:

1. Create a [ZipArchive](#) class instance.
2. Call its [ZipArchive.AddFile](#) method for each selected file. The method returns an object of the [ZipFileItem](#) type.
3. Specify the [ZipArchive.Password](#) and [ZipItem.EncryptionType](#) properties for each item.
4. Call the [ZipArchive.Save](#) method to create an archive and save it to a specified location.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ProtectPassword() {
    string[] sourcefiles = this.sourceFiles;
    string password = "123";
    using (ZipArchive archive = new ZipArchive()) {
        foreach (string file in sourceFiles) {
            ZipFileItem zipFI = archive.AddFile(file, "/");
            zipFI.EncryptionType = EncryptionType.Aes128;
            zipFI.Password = password;
        }
        archive.Save("Documents\\ProtectPassword.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ProtectPassword()
    Dim sourcefiles() As String = Me.sourceFiles
    Dim password As String = "123"
    Using archive As New ZipArchive()
        For Each file As String In Me.sourceFiles
            Dim zipFI As ZipFileItem = archive.AddFile(file, "/")
            zipFI.EncryptionType = EncryptionType.Aes128
            zipFI.Password = password
        Next file
        archive.Save("Documents\\ProtectPassword.zip")
    End Using
End Sub
```

How to: Add Comment to File

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Add Comment to File](#)

To add a comment to a file in an archive, do the following:

1. Create a [ZipArchive](#) class instance.
2. Call the [ZipArchive.AddFile](#) method for each selected file. This method returns an object of the [ZipFileItem](#) type.
3. Assign a text string to the [ZipItem.Comment](#) property.
4. Call the [ZipArchive.Save](#) method to create an archive and save it to a specified location.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example adds a text comment to each zip item in the archive that indicates the person who is currently logged on.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveWithComment() {
    string path = this.startupPath;
    using (ZipArchive archive = new ZipArchive()) {
        foreach (string file in System.IO.Directory.EnumerateFiles(path)) {
            ZipFileItem zipFI = archive.AddFile(file, "/");
            zipFI.Comment = "Archived by " + Environment.UserName;
        }
        archive.Save("Documents\\ArchiveWithComment.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveWithComment()
    Dim path As String = Me.startupPath
    Using archive As New ZipArchive()
        For Each file As String In System.IO.Directory.EnumerateFiles(path)
            Dim zipFI As ZipFileItem = archive.AddFile(file, "/")
            zipFI.Comment = "Archived by " & Environment.UserName
        Next file
        archive.Save("Documents\\ArchiveWithComment.zip")
    End Using
End Sub
```

How to: Compress .NET Stream

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Compress .NET Stream](#)

To compress a .NET stream, do the following:

1. Create a [ZipArchive](#) class instance.
2. Call its [ZipArchive.AddStream](#) method and specify a source stream.
3. Call the proper [ZipArchive.Save](#) method overload to create an archive and save it to a stream.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveStream() {
    using (Stream myStream = new MemoryStream(System.Text.Encoding.UTF8.GetBytes("DevExpress")))
        using (Stream myZippedStream = new FileStream("Documents\\ArchiveStream.zip", System.IO.FileMode.Create))
            using (ZipArchive archive = new ZipArchive()) {
                archive.AddStream("myStream", myStream);
                archive.Save(myZippedStream);
            }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveStream()
    Using myStream As Stream = New MemoryStream(System.Text.Encoding.UTF8.GetBytes("DevExpress"))
        Using myZippedStream As Stream = New FileStream("Documents\\ArchiveStream.zip", FileMode.Create)
            Using archive As New ZipArchive()
                archive.AddStream("myStream", myStream)
                archive.Save(myZippedStream)
            End Using
        End Using
    End Using
End Sub
```

How to: Compress Byte Array

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Compress Byte Array](#)

To compress a byte array, do the following:

1. Create a [ZipArchive](#) class instance.
2. Call its [ZipArchive.AddByteArray](#) method to specify a byte array to compress.
3. Call the [ZipArchive.Save](#) method to create an archive and save it to a stream.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This code snippet adds a byte array to an archive as an item with the name "myByteArray" and outputs zipped data to the stream.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveByteArray() {
    byte[] myByteArray = Enumerable.Repeat((byte)0x78, 10000).ToArray();
    using (Stream myZippedStream = new FileStream("Documents\\ArchiveByteArray.zip", FileMode.Create))
        using (ZipArchive archive = new ZipArchive()) {
            archive.AddByteArray("myByteArray", myByteArray);
            archive.Save(myZippedStream);
        }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveByteArray()
    Dim myByteArray() As Byte = Enumerable.Repeat(CByte(&H78), 10000)
    Using myZippedStream As Stream = New FileStream("Documents\\ArchiveByteArray.zip", FileMode.Create)
        Using archive As New ZipArchive()
            archive.AddByteArray("myByteArray", myByteArray)
            archive.Save(myZippedStream)
        End Using
    End Using
End Sub
```

How to: Unzip an Archive

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Unzip an Archive](#)

1. Call the static [ZipArchive.Read](#) method of the `DevExpress.Compression.ZipArchive` class. It has two overloads, so you can specify a compressed stream or a path to archive a file as the method's argument. This method returns a [ZipArchive](#) instance that provides access to a collection of [ZipItem](#) elements using indexed notation.
2. An archive item can be accessed using its archive path or index.
3. You can specify the [ZipArchive.Password](#) property value, if required.
4. For each `ZipItem` call the [ZipItem.Extract](#) method. It has two overloads, so you can either specify a destination stream or a destination file path.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example illustrates how to load the zip file and extract it to the specified directory. If the target directory is not specified, the current directory is the target.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void UnzipArchive() {
    string pathToZipArchive = "Documents\\Example.zip";
    string pathToExtract = "Documents\\!Extracted";
    using (ZipArchive archive = ZipArchive.Read(pathToZipArchive)) {
        foreach (ZipItem item in archive) {
            item.Extract(pathToExtract);
        }
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub UnzipArchive()
    Dim pathToZipArchive As String = "Documents\\Example.zip"
    Dim pathToExtract As String = "Documents\\!Extracted"
    Using archive As ZipArchive = ZipArchive.Read(pathToZipArchive)
        For Each item As ZipItem In archive
            item.Extract(pathToExtract)
        Next item
    End Using
End Sub
```

How to: Resolve File Conflicts when Unzipping

[Office File API](#) > [Zip Compression and Archive API](#) > [Examples](#) > [How to: Resolve File Conflicts when Unzipping](#)

To implement conflict resolution for file names when decompressing an archive, do the following:

1. Call the static [ZipArchive.Read](#) method of the `DevExpress.Compression.ZipArchive` class. It has two overloads, so you can specify a compressed stream or a path, to archive a file as the argument of the method. The method returns a [ZipArchive](#) instance that provides access to a collection of [ZipItem](#) elements using indexed notation.
2. Set the [ZipArchiveOptionsBehavior.AllowFileOverwrite](#) mode to [AllowFileOverwriteMode.Custom](#).
3. Subscribe to the [ZipArchive.AllowFileOverwrite](#) event. This event is raised for each `ZipItem` that conflicts with the existing file. It allows you to decide whether to overwrite an existing file with the same name or to skip it and proceed to the next item.
4. Call the [ZipItem.Extract](#) method for each `ZipItem`.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example illustrates how to handle a file name conflict when files are extracted from archive. If a file with the same name exists, the [ZipArchive.AllowFileOverwrite](#) event occurs. You can handle this event and determine that if the file in the folder is newer than the zip item, the file in the folder should not be overwritten.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void UnzipArchiveConflict() {
    string pathToZipArchive = "Documents\\Example.zip";
    string pathToExtract = "Documents\\!Extracted";
    using (ZipArchive archive = ZipArchive.Read(pathToZipArchive)) {
        archive.OptionsBehavior.AllowFileOverwrite = AllowFileOverwriteMode.Custom;
        archive.AllowFileOverwrite += archive_AllowFileOverwrite;
        foreach (ZipItem item in archive) {
            item.Extract(pathToExtract);
        }
    }
}

private void archive_AllowFileOverwrite(object sender, AllowFileOverwriteEventArgs e) {
    FileInfo fi = new FileInfo(e.TargetFilePath);
    if (e.ZipItem.LastWriteTime < fi.LastWriteTime) e.Cancel = true;
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression
Public Sub UnzipArchiveConflict()
    Dim pathToZipArchive As String = "Documents\Example.zip"
    Dim pathToExtract As String = "Documents\!Extracted"
    Using archive As ZipArchive = ZipArchive.Read(pathToZipArchive)
        archive.OptionsBehavior.AllowFileOverwrite = AllowFileOverwrite
        AddHandler archive.AllowFileOverwrite, AddressOf archive_A
        For Each item As ZipItem In archive
            item.Extract(pathToExtract)
        Next item
    End Using
End Sub
Private Sub archive_A-AllowFileOverwrite(ByVal sender As Object, ByVal e As EventArgs)
    Dim fi As New FileInfo(e.TargetFilePath)
    If e.ZipItem.LastWriteTime < fi.LastWriteTime Then
        e.Cancel = True
    End If
End Sub
End Sub
```

Barcode Generation API

[Office File API](#) > [Barcode Generation API](#)

The **Barcode Generation API** allows you to create barcode images in your .NET application. The library supports the majority of industry standard barcodes. It allows you to set options specific to each code type, and generate an image for use in your application or an image to insert into a document.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this component or library in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Main Features

- [Supported Bar Codes](#)
- [Bar Code Options](#)

Example

- [Getting Started](#)

Getting Started

[Office File API](#) > [Barcode Generation API](#) > [Getting Started](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

To get started with a Barcode Generation API, perform the following steps.

1. Create a new Windows Forms Application project.
2. Add reference to the **DevExpress.Docs.v18.1.dll** and the **DevExpress.Data.v18.1.dll** assemblies.
3. Drag a Button from the Toolbox and drop it on the form. Drag a PictureBox from the Toolbox and drop it on the form.
4. Insert the code listed below into the method that handles the Click button event.

C#

```
using DevExpress.BarCodes;
//...
this.pictureBox1.Image = null;
BarCode barCode = new BarCode();
barCode.Symbology = Symbology.QRCode;
barCode.CodeText = "http://www.devexpress.com";
barCode.BackColor = Color.White;
barCode.ForeColor = Color.Black;
barCode.RotationAngle = 0;
barCode.CodeBinaryData = Encoding.Default.GetBytes(barCode.CodeText);
barCode.Options.QRCode.CompactionMode = QRCodeCompactionMode.Byte;
barCode.Options.QRCode.ErrorLevel = QRCodeErrorLevel.Q;
barCode.Options.QRCode.ShowCodeText = false;
barCode.DpiX = 72;
barCode.DpiY = 72;
this.pictureBox1.Image = barCode.BarCodeImage;
pictureBox1.Size = pictureBox1.Image.Size;
```

Visual Basic

```
Imports DevExpress.BarCodes
'...
Me.pictureBox1.Image = Nothing
Dim barCode As New BarCode()
barCode.Symbology = Symbology.QRCode
barCode.CodeText = "http://www.devexpress.com"
barCode.BackColor = Color.White
barCode.ForeColor = Color.Black
barCode.RotationAngle = 0
barCode.CodeBinaryData = Encoding.Default.GetBytes(barCode.CodeText)
barCode.Options.QRCode.CompactionMode = QRCodeCompactionMode.Byte
barCode.Options.QRCode.ErrorLevel = QRCodeErrorLevel.Q
barCode.Options.QRCode.ShowCodeText = False
barCode.DpiX = 72
barCode.DpiY = 72
Me.pictureBox1.Image = barCode.BarCodeImage
pictureBox1.Size = pictureBox1.Image.Size
```

5. Run the project and click the button. See the resulting QR barcode in the PictureBox control.

See Also

[BarCode](#)
[Symbology](#)
[QRCodeOptions](#)
[Training Videos](#)

Concepts

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#)

The topics in this section cover the main functionality of the Barcode Library.

- [Bar Code Types](#)
- [Bar Code Options](#)
- [Bar Code Recognition Specifics](#)

Bar Code Types

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#)

The following bar code types are implemented in the Barcode Generation API:

1D Bar Codes

- [Codabar](#)
- [Code 11 \(USD-8\)](#)
- [Code 128](#)
- [Code 39 \(USD-3\)](#)
- [Code 39 Extended](#)
- [Code 93](#)
- [Code 93 Extended](#)
- [EAN 8](#)
- [EAN 13](#)
- [EAN-128 \(UCC\)](#)
- [Industrial 2 of 5](#)
- [Interleaved 2 of 5](#)
- [Matrix 2 of 5](#)
- [MSI - Plessey](#)
- [PostNet](#)
- [UPC Supplemental 2](#)
- [UPC Supplemental 5](#)
- [UPC-A](#)
- [UPC-E0](#)
- [UPC-E1](#)
- [GS1 DataBar](#)
- [UPC Shipping Container Symbol \(ITF-14\)](#)

2D Bar Codes

- [Data Matrix \(ECC200\)](#)
- [GS1- Data Matrix](#)
- [Intelligent Mail](#)
- [PDF417](#)
- [QR Code](#)

Codabar

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Codabar](#)

Short Description

The Codabar was developed in 1972 by Pitney Bowes, Inc. It is a discrete, self-checking symbology that may encode 16 different characters, plus an additional 4 start/stop characters. This symbology is used by U.S. blood banks, photo labs, and on FedEx air bills.



See Also

[BarCode](#)

[Symbology](#)

[CodabarOptions](#)

Code 11 (USD-8)

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Code 11 \(USD-8\)](#)

Short Description

Code 11, also known as USD-8, was developed as a high-density numerical-only symbology. It's used primarily in labeling telecommunications equipment. The symbology is discrete and is able to encode the numbers **0** through to **9**, the dash symbol (-), and start/stop characters.



See Also

[BarCode](#)
[Symbology](#)
[Code11Options](#)

Code 128

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Code 128](#)

Short Description

Code 128 is a very effective, high-density symbology which permits the encoding of alphanumeric data. The symbology includes a checksum digit for verification, and the barcode can also be verified character-by-character, allowing the parity of each data byte to be verified. This symbology has been widely implemented in many applications where a relatively large amount of data must be encoded in a relatively small amount of space. It's specific structure also allows numerical data to be encoded at, effectively, double-density.



See Also

[BarCode](#)
[Symbology](#)
[Code128Options](#)

Code 39 (USD-3)

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Code 39 \(USD-3\)](#)

Short Description

Code 39, the first alpha-numeric symbology to be developed, is still widely used, particularly in non-retail environments. It is the standard bar code used by the United States Department of Defense, and is also used by the Health Industry Bar Code Council (HIBCC). Code 39 is also known as "3 of 9 Code" and "USD-3".



See Also

[BarCode](#)
[Symbology](#)
[Code39Options](#)

Code 39 Extended

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Code 39 Extended](#)

Short Description

It is possible, using the Code 39's "Full ASCII Mode" to encode all 128 ASCII characters. This is accomplished by using the \$, /, %, and + symbols as "shift" characters. These characters combined with the single character that follows indicate which Full ASCII character is to be used.



See Also

[BarCode](#)

[Symbology](#)

[Code39ExtendedOptions](#)

Code 93

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Code 93](#)

Short Description

Code 93 was designed to supplement and improve upon Code 39. Code 93 is similar in that, like Code 39, can represent the full ASCII character set by using combinations of 2 characters. It differs in that Code 93 is a continuous symbology and produces denser code. It also encodes 47 characters compared to Code 39's 43 characters.



See Also

[BarCode](#)
[Symbology](#)
[Code93Options](#)

Code 93 Extended

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Code 93 Extended](#)

Short Description

It is possible, using Code 93's "Full ASCII Mode" to encode all 128 ASCII characters. This is accomplished by using the (\$), (/), (%), and (+) symbols as "shift" characters. These characters combined with the single character that follows indicate which Full ASCII character is to be used.



See Also

[BarCode](#)
[Symbolology](#)
[Code93ExtendedOptions](#)

EAN 13

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [EAN 13](#)

Short Description

EAN-13, based upon the UPC-A standard, was implemented by the International Article Numbering Association (EAN) in Europe. Now, the GS1 organization is responsible for the maintenance of barcode standards (refer to [GS1 Homepage](#) for more information.)



The EAN-13 barcode contains 13 digits, no letters or other characters. The first two or three digits represents the country. The leading zero actually signifies the USA, and UPC-A coding. The last digit is the "check digit", the checksum. The check digit is calculated using the first twelve figures when the barcode is constructed. So, for the correct EAN-13 code, you should specify only the first 12 digits.

See Also

[BarCode](#)
[Symbolology](#)
[EAN13Options](#)

EAN 8

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [EAN 8](#)

Short Description

EAN-8 is the EAN equivalent of UPC-E in the sense that it provides a "short" barcode for small packages.



See Also

[BarCode](#)
[Symbology](#)
[EAN8Options](#)

Data Matrix (ECC200)

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Data Matrix \(ECC200\)](#)

Short Description

Data Matrix code (**ISO/IEC 16022** international standard) is a two-dimensional matrix barcode consisting of black and white "cells" arranged in a rectangular pattern. The information to be encoded can be text or raw data. Every Data Matrix is composed of two solid adjacent borders in an "L" shape (called the "finder pattern"), and two other borders consisting of alternating dark and light cells or modules (called the "timing pattern"). Within these borders are rows and columns of cells that encode information. The finder pattern is used to locate and orient the symbol, while the timing pattern provides a count of the number of rows and columns in the symbol.



See Also

[BarCode](#)
[Symbology](#)
[DataMatrixOptions](#)

GS1- Data Matrix

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [GS1- Data Matrix](#)

Short Description

The **GS1 Data Matrix** uses a special start combination to differentiate the GS1 DataMatrix symbol from other Data Matrix ECC 200 symbols. This is achieved by using the Function 1 Symbol Character (FNC1) in the first position of the encoded data. It enables scanners to process the information according to the GS1 System Rules.



See Also

[BarCode](#)
[Symbology](#)
[DataMatrixGS1Options](#)

EAN-128 (UCC)

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [EAN-128 \(UCC\)](#)

Short Description

GS1-128 (EAN-128) was developed to provide a worldwide format and standard for exchanging common data between companies. While other barcodes simply encode data with no respect for what the data represents, **GS1-128** encodes data and encodes what that data represents.



See Also

[BarCode](#)
[Symbology](#)
[EAN128Options](#)

Industrial 2 of 5

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Industrial 2 of 5](#)

Short Description

Standard 2 of 5 is a low-density numerical symbology that was introduced in the 1960s. It has been used in the photofinishing and warehouse sorting industries, as well as to sequentially number airline tickets.



See Also

[BarCode](#)
[Symbology](#)
[Industrial2of5Options](#)

Intelligent Mail

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Intelligent Mail](#)

Short Description

The **Intelligent Mail** (IM) code is a 65-bar code for use on mail in the United States. This barcode is intended to provide greater information and functionality than its predecessors POSTNET and PLANET. Intelligent Mail barcode has also been referred to as **One Code Solution** and **4-State Customer** barcode abbreviated **4CB**, **4-CB** or **USPS4CB**.



See Also

[BarCode](#)
[Symbology](#)
[IntelligentMailOptions](#)

Interleaved 2 of 5

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Interleaved 2 of 5](#)

Short Description

Interleaved 2 of 5 is a higher-density numerical symbology based upon the Standard 2 of 5 symbology. It is used primarily in the distribution and warehouse industry.



See Also

[BarCode](#)
[Symbology](#)
[Interleaved2of5Options](#)

MSI - Plessey

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [MSI - Plessey](#)

Short Description

MSI was developed by the MSI Data Corporation, based on the original Plessey Code. MSI, also known as Modified Plessey, is used primarily to mark retail shelves for inventory control. MSI is a continuous, non-self-checking symbology. While the length of an MSI barcode can be of any length, a given application usually implements a fixed-length code.



See Also

[BarCode](#)
[Symbology](#)
[CodeMSIOptions](#)

Matrix 2 of 5

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [Matrix 2 of 5](#)

Short Description

Matrix 2 of 5 is a linear 1D barcode. Matrix 2 of 5 is a self-checking numerical-only barcode. Unlike the Interleaved 2 of 5, all of the information is encoded in the bars; the spaces are of a fixed width and used only to separate the bars. Matrix 2 of 5 is used primarily for warehouse sorting, photo finishing, and airline ticket marking.



See Also

[BarCode](#)
[Symbology](#)
[Matrix2of5Options](#)

PDF417

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [PDF417](#)

Short Description

PDF417 (Portable Data File) is a stacked linear two-dimensional barcode used in a variety of applications; primarily transport, postal, identification card and inventory management. It was invented by Ynjiun Wang at Symbol Technologies in 1991, and has spawned an Open Source decoder project together with an Open Source encoder. The **PDF417** barcode is also called a **symbol** barcode and usually consists of **3** to **90** rows, each of which is like a small linear bar code. For more information on this symbology, refer to its [Official Standard](#) web page.



See Also

[BarCode](#)
[Symbology](#)
[PDF417Options](#)

PostNet

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [PostNet](#)

Short Description

PostNet was developed by the United States Postal Service (USPS) to allow faster mail sorting and routing. PostNet codes are the familiar and unusual looking barcodes often printed on envelopes and business return mail. Unlike most other barcodes, in which data is encoded in the width of the bars and spaces, PostNet actually encodes data in the height of the bars. That's why all the bars are of the same width, but not the same height.



See Also

[BarCode](#)
[Symbology](#)
[PostNetOptions](#)

QR Code

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [QR Code](#)

Short Description

A **QR Code** (**QR** is the abbreviation for **Quick Response**) is a two-dimensional code, readable by QR scanners, mobile phones with a camera, and smartphones. QR Code can encode textual, numeric and binary data.



See Also

[BarCode](#)

[Symbology](#)

[QRCodeOptions](#)

[QRCodeOptions.Version](#)

UPC Supplemental 2

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [UPC Supplemental 2](#)

Short Description

2-digit supplemental barcodes should only be used with magazines, newspapers and other periodicals. The 2-digit supplement represents the issue number of the magazine. This is useful so that the product code itself (contained in the main barcode) is constant for the magazine, so that each issue of the magazine doesn't have to have its own unique barcode. Nevertheless, the 2-digit supplement can be used to track which issue of the magazine is being sold, for example, for sales analysis or restocking purposes.



See Also

[BarCode](#)
[Symbology](#)
[UPCSupplemental2Options](#)

UPC Supplemental 5

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [UPC Supplemental 5](#)

Short Description

5-digit supplemental barcodes are used on books to indicate the suggested retail price.



See Also

[BarCode](#)

[Symbology](#)

[UPCSupplemental5Options](#)

UPC-A

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [UPC-A](#)

Short Description

The "UPC-A barcode" is by far the most common and well-known symbology, especially in the United States. A UPC-A barcode is the barcode you will find on virtually every consumer item on the shelves of your local supermarket, as well as books, magazines, and newspapers. It is called simply, a "UPC barcode" or "UPC Symbol."



The UPC-A barcode contains 12 digits, no letters or other characters. The first digit is the prefix signifying the product type. The last digit is the "check digit". The check digit is calculated using first eleven figures when the barcode is constructed. So, for the correct UPC-A you should specify only the first 11 digits.

The recommended dimensions are shown in the picture. The standard allows magnification up to 200 %, and reduction of up to 80 % of the recommended size.

There should be two quiet zones before and after the barcode. They provide reliable operation of the barcode scanner. The quiet zone recommended length is 2.97 mm for the barcode of standard width and height.

See Also

[BarCode](#)
[Symbology](#)
[UPCAOptions](#)

UPC-E0

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [UPC-E0](#)

Short Description

UPC-E is a variation of UPC-A which allows for a more compact barcode by eliminating "extra" zeros. Since the resulting UPC-E barcode is about half the size as an UPC-A barcode, UPC-E is generally used on products with very small packaging, where a full UPC-A barcode couldn't reasonably fit.

The UPC-E0 is a kind of UPC-E code with the number system set to 0. In the human readable string of the bar code the first digit signifies the number system (always 0 for this code type), and the last digit is the check digit of the original UPC-A code. So, in the example below, the original UPC-A code is "04210000526". We should remove the leading zero when assigning the string to the control's property, since the code format itself implies its presence. The checksum digit (4) is calculated automatically, and the symbology algorithm transforms the rest of the numeral string. The result is 425261, and it is encoded along with the number system prefix and the check digit into the scanner-readable form.

Not every UPC-A code can be transformed into the UPC-E0. It must meet special requirements, and you should refer to [UPC-E Symbology page](#) for more information.



See Also

[BarCode Symbology](#)
[UPCE0Options](#)

UPC-E1

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [UPC-E1](#)

Short Description

UPC-E is a kind of UPC-A, which allows a more compact barcode by eliminating "extra" zeros. Since the resulting UPC-E barcode is about half the size of the UPC-A barcode, UPC-E is generally used on products with a very small packaging where a full UPC-A barcode doesn't fit.

The UPC-E1 is a variation of UPC-E code with the number system set to "1". In the human readable string of the bar code the first digit signifies the number system (always 1 for this code type), the last digit is the check digit of the original UPC-A code. So, in the example below, the original UPC-A code is "14210000526". We should remove the leading "1" when assigning the string to the control's property, since the code format itself implies its presence. The checksum digit (1) is calculated automatically, and the symbology algorithm transforms the rest of the numeral string. The result is 425261, and it is encoded along with the number system prefix and the check digit into the scanner-readable form.

Not every UPC-A code can be transformed into the UPC-E1. It must meet special requirements, and you may refer to [UPC-E Symbology page](#) for more information.



Since the number system "1" is not used in regular UPC-A codes, the UPC-E1 symbology use is uncommon.

See Also

[BarCode](#)
[Symbology](#)
[UPCE1Options](#)

GS1 DataBar

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [GS1 DataBar](#)

Short Description

The **GS1 DataBar** bar code is based on a family of symbols often used in the **GS1 DataBar Coupon** (coupon codes commonly used in retail).

These bar codes can encode up to **14** digits, which makes them suitable for **GTIN 8, 12, 13** and **14**.

GS1 DataBar Expanded and **GS1 DataBar Expanded Stacked** can encode up to **74** numeric or **41** alphanumeric characters, and provide the capability to utilize all **GS1 Application Identifiers** (e.g., expiration date, batch and serial number). These bar codes are often used in manufacturer coupons.



See Also

[BarCode](#)
[Symbology](#)
[BarCodeOptions.DataBar](#)
[DataBarOptions](#)
[DataBarType](#)

UPC Shipping Container Symbol (ITF-14)

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Types](#) > [UPC Shipping Container Symbol \(ITF-14\)](#)

Short Description

The **UPC Shipping Container Symbol (ITF-14)** bar code is used to mark packaging materials that contain products labeled with a **UPC** or **EAN** product identification number.

This bar code provides a **GS1** implementation of an **Interleaved 2 of 5** bar code for encoding a **Global Trade Item Number** (an identifier for trade items developed by **GS1**). This bar code always uses a total of **14** digits.

The thick black border around the symbol (the **Bearer Bar**) is intended to improve bar code reading reliability.



See Also

[BarCode](#)
[Symbology](#)
[BarCodeOptions.ITF14](#)
[Interleaved2of5Options](#)


Bar Code Options

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Options](#)

The library allows you to set options specific for each code type and generate an image for use in your application or for inserting into a document.

When generating an image, the library allows you to set its orientation, color, quality, and also allows you to specify whether or not an image displays a bar code text. You can include top and bottom captions with arbitrary text formatted using font, color and alignment options.

EAN8



EAN-8 is the EAN equivalent of UPC-E in the sense that it provides a "short" barcode for small packages.

BarCode properties

☒ Show text

Text: 01234565

Vert Align: Center

Horz Align: Center

Font Name: Sylfaen

Angle: 45

Fore Color: 0, 128, 0

Back Color: White

Top Caption

☒ Show Caption

Text: top caption

Alignment: Center

Fore Color: 255, 128, 0

Font Name:

Bottom Caption

☒ Show Caption

Text: bottom caption

Alignment: Center

Fore Color: 255, 128, 0

Font Name:

Options common for all bar codes ([BarCode.BackColor](#), [BarCode.ForeColor](#), [BarCode.RotationAngle](#) etc.) are accessible via the properties of the [BarCode](#) object.

Options specific to a certain symbology are accessible via the properties of a [BarCodeOptions](#) object exposed by the [BarCode.Options](#) property.

See Also

[BarCode](#)
[BarCodeOptions](#)

Bar Code Recognition Specifics

[Office File API](#) > [Barcode Generation API](#) > [Concepts](#) > [Bar Code Recognition Specifics](#)

This document describes the main specifics of bar code recognition and how to resolve the most frequently encountered issues when working with bar codes.

This topic consists of the following sections.

- [Choose an Appropriate Bar Code Type](#)
- [Insert the Function Code One Character \(FNC1\) or the Application Identifier into a Bar Code](#)
- [Specify the Bar Code Resolution on Export to Third-Party Formats](#)
- [Use Bar Codes in Point-of-Sale \(POS\) Systems](#)
- [Common Issues](#)

Choose an Appropriate Bar Code Type

Selecting an appropriate bar code type (symbology) depends on your specific business requirements and the applied industrial standards.

In general, we recommend that you consider using [Bar Code 2 of 5 Interleaved](#) for encoding digits and [Bar Code 39](#) for encoding the full range of ASCII characters.

For a list of supported bar codes, see [Bar Code Types](#).

Insert the Function Code One Character (FNC1) or the Application Identifier into a Bar Code

Some encodings enable you to insert a special **FNC1** character for separating application identifiers from the rest of the bar code.

According to the **GS1** specification, the **FNC1** character is always inserted at the first position of the encoded data. Other identifiers can be inserted manually using the default "#" character.

Although you can use any ASCII character as the **FNC1** placeholder, it will not be a part of the encoded data as it does not have any direct ASCII representation.

Note

For the [Code 128](#) symbology, only **FNC1** characters are currently supported. At present, there is no way to define **FNC2 - 4** characters for this bar code in DevExpress products.

For the list of the available application identifiers, refer to the official documentation at www.gs1.org.

Specify the Bar Code Resolution on Export to Different Formats

At present, only export to PDF using the [BarCode.ExportToPdf](#) method preserves the original bar code in its vector form. Export to other formats will keep only the rasterized version of a bar code (with the default DPI set to **96**).

Use Bar Codes in Point-of-Sale (POS) Systems

Bar Code Library is built on top of the .NET Framework that does not provide an out-of-the-box support for any matrix and thermal printers.

To access the internal printer fonts or achieve the fastest printing speed possible, use the native approach suggested by the printer manufacturer. Typically, a manufacturer would provide a special series of control commands that should be directly sent to a printer's port.

Common Issues

This document section provides solutions to the most common issues that you may encounter when creating bar codes.

- **The bar code is too "dense"**

The more information you wish to encode, the more bars should be drawn and the larger the bar code should become.

The [BarCode.Module](#) property specifies the width of the narrowest bar in a bar code. Although you can set this property to a very small [Double](#) value, the actual value is determined by the maximum resolution of your bar code printer device.

Alternatively, consider using the [BarCode.AutoSize](#) option to automatically calculate the optimal bar size based on the current bar code dimensions.

 **Note**

When bar codes are "dense" and you are manually specifying the Module value, make sure that multiplying this value by the bar code printer resolution results in an integer number. Otherwise, rounding errors may occur on calculating the resulting bar width.

For example, when the Module is set to **0.015** inches and the printer resolution is **300** DPI, their product equals **4.5**, which may be rounded to **4** or **5** pixels for different bars and result in bar code recognition errors. In this case, the Module property should be set to **0.01333** (to make the bar width equal to **4** pixels) or to **0.01667** (to make the bar width equal to **5** pixels).

- **The bar code is correctly displayed on the preview but it is not scanned**

Make sure that your scanner has been correctly set up to be able to recognize a specific kind of a bar code. If you are not certain about how to operate the scanner properly, please refer to its product manual.

Avoid scanning bar codes from the monitor screen (e.g., using an application installed on your smartphone), because the screen DPI may not be sufficient to effectively recognize each particular bar.

- **The bar code is correctly displayed on the preview but it is scanned incorrectly**

The cause for this problem may be an encoding issue specific to the "binary" input mode.

By default, the .NET Framework and all [String](#) objects leverage the **UTF-16** encoding (see [Character Encoding in the .NET Framework](#)). However, your scanner device may use a different encoding model or even a codepage (i.e., a specific table that maps abstract values to real human-understandable characters). For additional information on this subject, please refer to the specification of your scanner device.


- **The "There are invalid characters in the text" error occurs**

Different bar code symbologies define different ranges of allowed characters under different character sets. To avoid this error, please check the bar code specification.

Unit Conversion API

[Office File API](#) > [Unit Conversion API](#)

The **Unit Conversion API** provides you with a set of extension methods for the **System.Double** data type. It enables you to easily perform conversions between different units of measurement and operate with quantity values (i.e., physical values expressed in units of measurement).

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this component or library in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

Main Features

The supported units of measurement are listed in the following table.

Physical Quantity	Unit of Measurement
Area	<ul style="list-style-type: none">AcreInternationalAcreStatuteAreHectareMorgenSquareAngstromSquareFootSquareInchSquareLightYearSquareMeterSquareMileSquareMileNauticalSquarePicaPointSquareYard
Distance	<ul style="list-style-type: none">AngstromEllInchFootLightYearMeterMileNauticalMileStatuteMileUSSurveyParsecPicaPicaPointYard
Energy	<ul style="list-style-type: none">BritishThermalUnitCalorieThermodynamicCalorieITElectronVoltErgFootPound

	<ul style="list-style-type: none">• HorsePowerHour• Joule• WattHour
Force	<ul style="list-style-type: none">• Dyne• Newton• Pond• Pound
Information	<ul style="list-style-type: none">• Bit• Byte
Magnetism	<ul style="list-style-type: none">• Tesla• Gauss
Mass	<ul style="list-style-type: none">• AtomicMassUnit• Gram• Grain• Hundredweight• HundredweightImperial• Ounce• Pound• Slug• Stone• Ton• TonImperial
Power	<ul style="list-style-type: none">• HorsePower• Watt
Pressure	<ul style="list-style-type: none">• Atmosphere• MmHg• Pascal• PoundPerSquareInch Torr
Speed	<ul style="list-style-type: none">• Knot• KnotAdmiralty• MetersPerHour• MetersPerSecond• MilesPerHour
Temperature	<ul style="list-style-type: none">• Celcius• Fahrenheit• Kelvin• Rankine• Reaumur
Time	<ul style="list-style-type: none">• Day• Hour

	<ul style="list-style-type: none">• Minute• Second• Year
Volume	<ul style="list-style-type: none">• Bushel• CubicAngstrom• CubicFoot• CubicInch• CubicLightYear• CubicMeter• CubicMile• CubicMileNautical• CubicPicaPoint• CubicYard• Cup• Gallon• GallonImperial• GrossRegisteredTon• Liter• MeasurementTon• OilBarrel• OunceFluid• Pint• PintImperial• Quart• QuartImperial• Tablespoon• Teaspoon• TeaspoonModern

Examples

- [Getting Started](#)
- [How to: Convert Between Metric and US Units of Measurement](#)

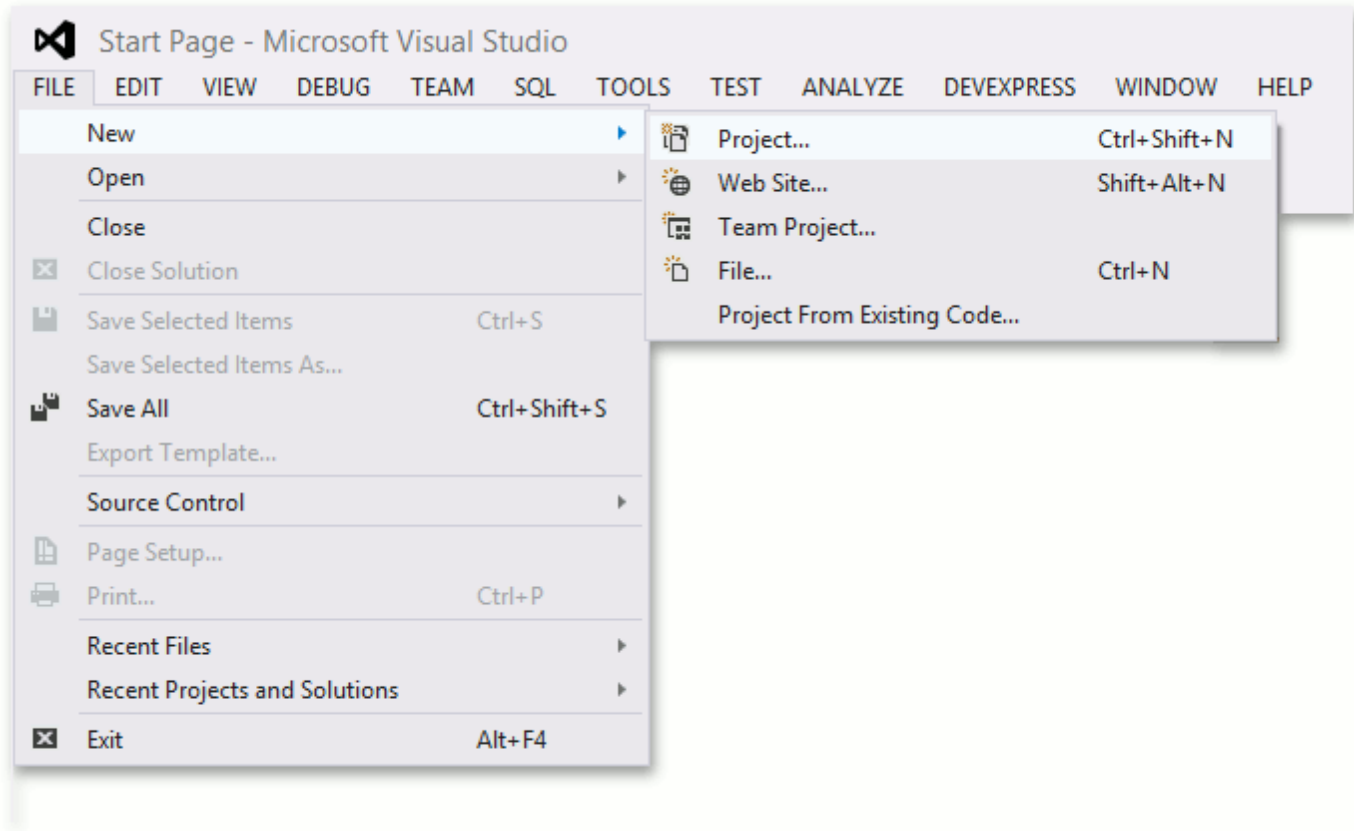
Getting Started

[Office File API](#) > [Unit Conversion API](#) > [Getting Started](#)

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

- To get started with the Unit Conversion API, perform the following steps. □
1. Start Microsoft Visual Studio and create a new Windows Forms Application project by selecting **FILE | New | Project** in the main menu. In the invoked **New Project** window, select **Windows Forms Application**, specify the name of the project and click **OK**.



2. On the Project menu, choose **Add Reference**. Alternatively, you can right-click your project in Solution Explorer and then click **Add Reference**. The **Reference Manager** dialog window is displayed. Select the **DevExpress.Docs.v18.1.dll** assembly and then click OK.
3. Drag a Button control from the Toolbox and drop it on the form.
4. Double click the button. In the button click handler, insert the code below.

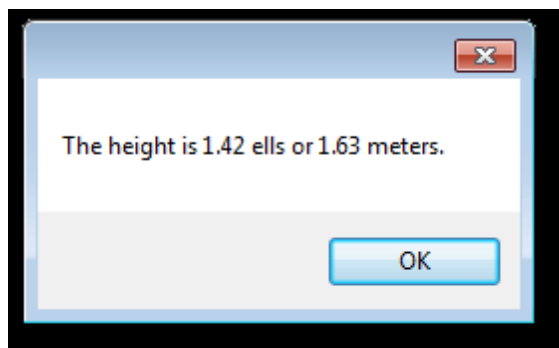
C#

```
using DevExpress.UnitConversion;
//...
// The height is 5'4".
QuantityValue<Distance> height = (5.0).Feet() + (4.0).Inches();
string s = String.Format("The height is {0} ells or {1} meters.",
    height.ToElls().Value.ToString("g3"), height.ToMeters().Value.ToString("g3"));
MessageBox.Show(s);
```

Visual Basic

```
Imports DevExpress.UnitConversion
'...
' The height is 5'4".
Private height As QuantityValue(Of Distance) = (5.0).Feet() + (4.0).Inches()
Private s As String = String.Format("The height is {0} ells or {1} meters.", _
    height.ToElls().Value.ToString("g3"), height.ToMeters().Value.ToString("g3"))
MessageBox.Show(s)
```

5. Run the project and execute the method. This will result in the display of the following message box.



See Also

[Training Videos](#)

Examples

[Office File API](#) > [Unit Conversion API](#) > [Examples](#)

This section provides a list of examples contained in this help.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

- [How to: Convert Between Metric Prefixes](#)
- [How to: Convert Between Metric and US Units of Measurement](#)

How to: Convert Between Metric Prefixes

[Office File API](#) > [Unit Conversion API](#) > [Examples](#) > [How to: Convert Between Metric Prefixes](#)

The **Unit Conversion API** includes a special converter that helps you convert between different metric prefixes. The following code illustrates how to display a pressure that is equal to 760 millimeters of mercury in hectopascals.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4921>.

C#

```
(Program.cs)
using DevExpress.UnitConversion;
using System.Diagnostics;

// The pressure value is set to 760 mmHg.
QuantityValue<Pressure> pressure = (760).MmHg();
// Since it is a quantity value, it should be transformed to a proper measurement unit
// to obtain a value for display or for comparison.
// The pressure is obtained in Pascals and then converted into hectoPascals.
MetricUnitsConverter prefixConverter = new MetricUnitsConverter();
double pressure_in_hPa = prefixConverter.Convert(pressure.ToPascals(), MetricPrefix.None, MetricPrefix.Hecto);
Debug.WriteLine(pressure in hPa);
```

Visual Basic

```
(Program.vb)
Imports DevExpress.UnitConversion
Imports System.Diagnostics

' The pressure value is set to 760 mmHg.
Dim pressure As QuantityValue(Of Pressure) = (760).MmHg()
' Since it is a quantity value, it should be transformed to a proper measurement unit
' to obtain a value for display or for comparison.
' The pressure is obtained in Pascals and then converted into hectoPascals.
Dim prefixConverter As New MetricUnitsConverter()
Dim pressure_in_hPa As Double = prefixConverter.Convert(pressure.ToPascals(), MetricPrefix.None, MetricPrefix.Hecto)
Debug.WriteLine(pressure in hPa)
```

How to: Convert Between Metric and US Units of Measurement

[Office File API](#) > [Unit Conversion API](#) > [Examples](#) > [How to: Convert Between Metric and US Units of Measurement](#)

This example illustrates the use of a [QuantityValue<T>](#) object and the Unit Conversion API converters for calculating the volume and the mass of the water in the aquarium tank; followed by a display of the results using US and metric units.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4686>.

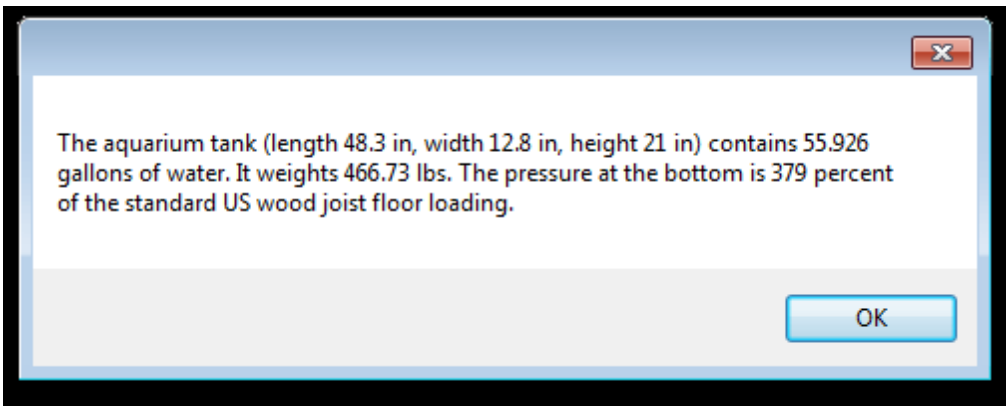
C#

```
(Form1.cs)
using DevExpress.UnitConversion;
//Quantity value is a value measured in a certain unit.
//You cannot mix different quantity values in a single expression.
//Instead, you can easily convert them as required and use the Value property to obtain numerical values.
QuantityValue<Distance> l = (48.25).Inches();
QuantityValue<Distance> w = (12.75).Inches();
QuantityValue<Distance> h = (21.0).Inches();
QuantityValue<Volume> vol = (l.ToMeters().Value * w.ToMeters().Value * h.ToMeters().Value).CubicMeters();
QuantityValue<Mass> m = (vol.ToLiters() * 1000).Value.Grams();
QuantityValue<Pressure> loading = (m.ToPounds() / (l * w).Value.SquareInches()).Value.PoundsPerSquareInch;
//US Standard uniform floor load is 0.2 psi.
double percentageUS = loading.ToPoundsPerSquareInch() / (0.2).PoundsPerSquareInch() * 100;
//British standard imposed load is 1500 Newtons per square meter.
double percentageEu = loading.ToPascals() / (1500.0).Pascals() * 100;
string textFormatUS = "The aquarium tank (length {0:g3} in, width {1:g3} in, height {2:g3} in) contains {3:g3} gallons. " +
    "It weights {4:g5} lbs. " +
    "The pressure at the bottom is {5:g3} percent of the standard US wood joist floor loading.";
string textUS = String.Format(textFormatUS, l.ToInches().Value, w.ToInches().Value,
    h.ToInches().Value, vol.ToGallons().Value, m.ToPounds().Value, percentageUS);
string textFormatEu = "The aquarium tank (length {0:g3} m, width {1:g3} m, height {2:g3} m) contains {3:g3} liters. " +
    "It weights {4:g5} kg. " +
    "The pressure at the bottom is {5:g3} percent of the standard UK wood joist floor loading.";
string textEu = String.Format(textFormatEu, l.ToMeters().Value, w.ToMeters().Value,
    h.ToMeters().Value, vol.ToLiters().Value, m.ToKilograms().Value, percentageEu);
string msg = radioButtonUS.Checked ? textUS : textEu;
MessageBox.Show(msg);
```

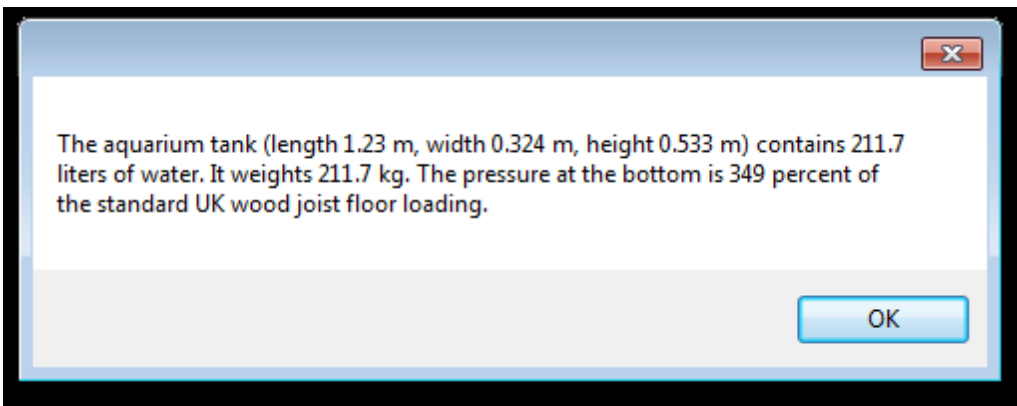
Visual Basic

```
(Form1.vb)
Imports DevExpress.UnitConversion
'Quantity value is a value measured in a certain unit.
'You cannot mix different quantity values in a single expression.
'Instead, you can easily convert them as required and use the Value property
Dim l As QuantityValue(Of Distance) = (48.25).Inches()
Dim w As QuantityValue(Of Distance) = (12.75).Inches()
Dim h As QuantityValue(Of Distance) = (21.0).Inches()
Dim vol As QuantityValue(Of Volume) = (l.ToMeters().Value * w.ToMeters().Value * h.ToMeters().Value).Liters()
Dim m As QuantityValue(Of Mass) = (vol.ToLiters() * 1000).Value.Grams()
Dim loading As QuantityValue(Of Pressure) = (m.ToPounds() / (l * w).Value).PoundsPerSquareInch()
'US Standard uniform floor load is 0.2 psi.
Dim percentageUS As Double = loading.ToPoundsPerSquareInch() / (0.2).PoundsPerSquareInch()
'British standard imposed load is 1500 Newtons per square meter.
Dim percentageEu As Double = loading.ToPascals() / (1500.0).Pascals() * 100
Dim textFormatUS As String = "The aquarium tank (length {0:g3} in, width {1:g3} in, height {2:g3} in) contains {3:g3} gallons of water. It weights {4:g3} lbs. The pressure at the bottom is {5:g3} percent of the standard US wood joist floor loading."
Dim textUS As String = String.Format(textFormatUS, l.ToInches().Value, w.ToInches().Value, h.ToInches().Value, vol.ToGallons().Value, m.ToPounds().Value, percentageUS)
Dim textFormatEu As String = "The aquarium tank (length {0:g3} m, width {1:g3} m, height {2:g3} m) contains {3:g3} liters of water. It weights {4:g3} kg. The pressure at the bottom is {5:g3} percent of the standard UK wood joist floor loading."
Dim textEu As String = String.Format(textFormatEu, l.ToMeters().Value, w.ToMeters().Value, h.ToMeters().Value, vol.ToLiters().Value, m.ToKilograms().Value, percentageEu)
Dim msg As String = If(radioButtonUS.Checked, textUS, textEu)
MessageBox.Show(msg)
```

For a US location, the output string looks like the following.



For a UK location, the output string looks like the following.



API Reference

DevExpress.BarCodes

Contains classes of the bar code library that allow you to generate bar code images with specific settings.

Classes

	Class	Description
	BarCode	A non-visual object that can generate a bar code for given data with the specified options.
	BarCodeCaption	Provides access to the bar code caption characteristics.
	BarCodeGeneratorOptions	Base class for options specific to a certain bar code symbology.
	BarCodeMargins	Contains values used to specify bar code margins.
	BarCodeOptions	Groups options that are specific to a certain symbology.
	BarCodePadding	Contains values used to specify bar code paddings.
	CodabarOptions	Contains options specific to this symbology (bar code type).
	Code11Options	Contains options specific to this symbology (bar code type).
	Code128Options	Contains options specific to this symbology (bar code type).
	Code39ExtendedOptions	Contains options specific to this symbology (bar code type).
	Code39Options	Contains options specific to this symbology (bar code type).
	Code93ExtendedOptions	Contains options specific to this symbology (bar code type).
	Code93Options	Contains options specific to this symbology (bar code type).
	CodeMSIOptions	Contains options specific to this symbology (bar code type).
	DataBarOptions	Contains options specific to this symbology (bar code type).
	DataMatrixGSIOptions	Contains options specific to this symbology (bar code type).
	DataMatrixOptions	Contains options specific to this symbology (bar code type).
	EAN128Options	Contains options specific to this symbology (bar code type).

	EAN13Options	Contains options specific to this symbology (bar code type).
	EAN8Options	Contains options specific to this symbology (bar code type).
	Industrial2of5Options	Contains options specific to this symbology (bar code type).
	IntelligentMailOptions	Contains options specific to this symbology (bar code type).
	Interleaved2of5Options	Contains options specific to this symbology (bar code type).
	Matrix2of5Options	Contains options specific to this symbology (bar code type).
	PDF417Options	Contains options specific to this symbology (bar code type).
	PostNetOptions	Contains options specific to this symbology (bar code type).
	QRCodeOptions	Contains options specific to this symbology (bar code type).
	UPCAOptions	Contains options specific to this symbology (bar code type).
	UPCE0Options	Contains options specific to this symbology (bar code type).
	UPCE1Options	Contains options specific to this symbology (bar code type).
	UPCSupplemental2Options	Contains options specific to this symbology (bar code type).
	UPCSupplemental5Options	Contains options specific to this symbology (bar code type).

Enumerations

	Enumeration	Description
	CodabarStartStopPair	Lists start/stop patterns used in bar codes.
	Code128CharacterSet	Lists character sets used in Code128 bar code symbology.
	DataBarType	Lists symbol types of the GS1 DataBar family.
	DataMatrixCompactionMode	Lists compaction modes used in DataMatrix bar code symbology.
	DataMatrixSize	Lists the available data matrix size options related to the Data Matrix (ECC200) bar code.
	PDF417CompactionMode	Lists compaction modes used in PDF417 symbology.
	PDF417ErrorLevel	Lists levels of error correction.

	QRCodeCompactionMode	Lists compaction modes for the QRCode bar code symbology.
	QRCodeErrorLevel	Lists error correction levels for the QRCode.
	QRCodeVersion	Lists values used to specify the QR Code version.
	Symbology	Lists available symbologies (bar code types)

Barcode Class

A non-visual object that can generate a bar code for given data with the specified options.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class Barcode Inherits Object, IDisposable,  
    IPrintable,  
    IBasePrintable
```

C#

```
public class Barcode : object, IDisposable,  
    IPrintable,  
    IBasePrintable
```

Remarks

Use the [CodeText](#) or the [CodeBinaryData](#) to set the data to be represented by a bar code. To choose the bar code type (symbology), specify the [Symbology](#) property. Note that each symbology has its own restrictions for the encoded data.

Symbology specifics are available via the [Options](#) property of a **Barcode** object.

Various visual options can be set using other **Barcode** properties.

The [BarcodeImage](#) property gets an image that is the bar code.

You can use the [Print](#), [PrintDialog](#) or the [ShowPrintPreview](#) methods to print the bar code.

The [ExportToPdf](#) method enables you to print a bar code to PDF.

Inheritance Hierarchy

[Object](#)

Barcode

See Also

[Barcode Members](#)


[DevExpress.BarCodes Namespace](#)

BarCode Members

A non-visual object that can generate a bar code for given data with the specified options.

The following tables list the members exposed by the [BarCode](#) type.

Public Constructors




	Name	Description
	BarCode	Initializes a new instance of the BarCode class with the default settings.


Public Properties

	Name	Description
	AutoSize	Allows you to create an image in a size that fits all the specified bar code options.
	BackColor	Gets or sets the background color of a bar code image.
	BarCodeImage	Gets the bar code image.
	BarHeight	Gets or sets the height of bars in a bar code.
	BorderColor	Gets or sets the border color of the bar code image.
	BorderWidth	Specifies the border width of the bar code image.
	BottomCaption	Provides access to characteristics of the bar code bottom caption.
	Code	Get the processed code text actually shown in the bar code.
	CodeBinaryData	Gets or sets the byte array to be coded into certain bar code types.
	CodeText	Gets or sets the text to be coded using the bar code symbology.
	CodeTextFont	Specifies the font used to display the bar code text.
	CodeTextHorizontalAlignment	Get or sets the horizontal alignment of the bar code text within the bar code image.
	CodeTextVerticalAlignment	Get or sets the vertical alignment of the bar code text within the bar code image.
	Dpi	Gets or sets the dpi value used to render the bar code.
	DpiX	Gets or sets the dpi value for the X-coordinate used to render the bar code.

	DpiY	Gets or sets the dpi value for the Y-coordinate used to render the bar code.
	ForeColor	Gets or sets the color of bars in the bar code image.
	ImageHeight	Gets or sets the height of the bar code image.
	ImageSize	Gets or sets the size of the bar code image.
	ImageWidth	Gets or sets the width of the resulting image.
	IsPrintingAvailable	Indicates whether the bar code can be printed and exported to PDF.
	Margins	Provides access to margin widths set for a bar code image.
	Module	Gets or sets the width of the narrowest bar or space in the bar code.
	Options	Provides access to options specific for various bar code types.
	Paddings	Gets the padding settings of a bar code image.
	RotationAngle	Gets or sets the number of degrees the bar code is rotated around the z-axis.
	ShowText	Indicates whether the code text is shown.
	Symbology	Gets or sets the bar code type (symbology).
	TextRenderingHint	Gets or sets the bar code text rendering quality.
	TopCaption	Provides access to characteristics of the bar code top caption.
	Unit	Gets or sets the unit of measurement for bar code settings.

Public Methods

	Name	Description
	Dispose	Resets the bar code image, disposes of the BarCode object and releases all the allocated resources.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current

		System.Object . (Inherited from Object)
	ExportToPdf	Overloaded. Exports the bar code image to the specified file path in PDF format.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Print	Prints the bar code to the default printer.
	PrintDialog	Invokes a dialog that enables you to select a printer and change print settings.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	Save	Overloaded. Saves the bar code image to the stream in a specified format.
	ShowPrintPreview	Invokes a dialog that enables you to preview the printout, print or save it in PDF format, or as an image file.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[BarCode Members](#)[DevExpress.BarCodes Namespace](#)

Barcode Constructor

Initializes a new instance of the [Barcode](#) class with the default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Barcode()
```

Remarks

Default bar code is a black-and-white bar code of the [EAN 13](#) symbology encoding the "0" symbol.

See Also

[Barcode Class](#)

[Barcode Members](#)


[DevExpress.BarCodes Namespace](#)

BarCode Properties

A non-visual object that can generate a bar code for given data with the specified options.

The following tables list the members exposed by the [BarCode](#) type.

Public Constructors




	Name	Description
	BarCode	Initializes a new instance of the BarCode class with the default settings.





Public Properties

	Name	Description
	AutoSize	Allows you to create an image in a size that fits all the specified bar code options.
	BackColor	Gets or sets the background color of a bar code image.
	BarCodeImage	Gets the bar code image.
	BarHeight	Gets or sets the height of bars in a bar code.
	BorderColor	Gets or sets the border color of the bar code image.
	BorderWidth	Specifies the border width of the bar code image.
	BottomCaption	Provides access to characteristics of the bar code bottom caption.
	Code	Get the processed code text actually shown in the bar code.
	CodeBinaryData	Gets or sets the byte array to be coded into certain bar code types.
	CodeText	Gets or sets the text to be coded using the bar code symbology.
	CodeTextFont	Specifies the font used to display the bar code text.
	CodeTextHorizontalAlignment	Get or sets the horizontal alignment of the bar code text within the bar code image.
	CodeTextVerticalAlignment	Get or sets the vertical alignment of the bar code text within the bar code image.
	Dpi	Gets or sets the dpi value used to render the bar code.
	DpiX	Gets or sets the dpi value for the X-coordinate used to render the bar code.

	DpiY	Gets or sets the dpi value for the Y-coordinate used to render the bar code.
	ForeColor	Gets or sets the color of bars in the bar code image.
	ImageHeight	Gets or sets the height of the bar code image.
	ImageSize	Gets or sets the size of the bar code image.
	ImageWidth	Gets or sets the width of the resulting image.
	IsPrintingAvailable	Indicates whether the bar code can be printed and exported to PDF.
	Margins	Provides access to margin widths set for a bar code image.
	Module	Gets or sets the width of the narrowest bar or space in the bar code.
	Options	Provides access to options specific for various bar code types.
	Paddings	Gets the padding settings of a bar code image.
	RotationAngle	Gets or sets the number of degrees the bar code is rotated around the z-axis.
	ShowText	Indicates whether the code text is shown.
	Symbology	Gets or sets the bar code type (symbology).
	TextRenderingHint	Gets or sets the bar code text rendering quality.
	TopCaption	Provides access to characteristics of the bar code top caption.
	Unit	Gets or sets the unit of measurement for bar code settings.

Public Methods

	Name	Description
	Dispose	Resets the bar code image, disposes of the BarCode object and releases all the allocated resources.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current

		System.Object . (Inherited from Object)
	ExportToPdf	Overloaded. Exports the bar code image to the specified file path in PDF format.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Print	Prints the bar code to the default printer.
	PrintDialog	Invokes a dialog that enables you to select a printer and change print settings.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	Save	Overloaded. Saves the bar code image to the stream in a specified format.
	ShowPrintPreview	Invokes a dialog that enables you to preview the printout, print or save it in PDF format, or as an image file.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[BarCode Members](#)[DevExpress.BarCodes Namespace](#)

AutoSize Property

Allows you to create an image in a size that fits all the specified bar code options.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property AutoSize As Boolean
```

C#

```
public bool AutoSize { get; set; }
```

Property Value

true, to create an image unrestricted to the specified size; otherwise, **false**.

Remarks

If the **AutoSize** is set to **true**, the [ImageWidth](#) and [ImageHeight](#) are not taken into account when an image is generated. You can get the image in a size that satisfies all the specified options.

If you set the **AutoSize** to **false**, the image will be clipped to the specified size. In this situation, you have to adjust the options which affect the image size ([Dpi](#), [TopCaption](#), [BottomCaption](#) etc.) manually to achieve the desired size.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

BackColor Property

Gets or sets the background color of a bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property BackColor As Color
```

C#

```
public Color BackColor { get; set; }
```

Property Value

A [System.Drawing.Color](#) object specifying the image background color.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

BarcodeImage Property

Gets the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property BarcodeImage As Bitmap
```

C#

```
public Bitmap BarcodeImage { get; }
```

Property Value

A [System.Drawing.Bitmap](#) object that is the bar code image.

Remarks

Use the **BarcodeImage** property to obtain the resulting bar code image. The bar code image is reset when any **Barcode** property is changed.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

BarHeight Property

Gets or sets the height of bars in a bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property BarHeight As Single
```

C#

```
public Single BarHeight { get; set; }
```

Property Value

A [System.Single](#) value that specifies the bar height in current units.

Remarks

The **BarHeight** enables you to adjust the bar code and maintain scanability. It allows reducing the height of the bar code without a reduction in width.

For **2D bar codes** the **BarHeight** property is not applicable.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

BorderColor Property

Gets or sets the border color of the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property BorderColor As Color
```

C#

```
public Color BorderColor { get; set; }
```

Property Value

A [System.Drawing.Color](#) object that is the border color.

Remarks

The **BorderColor** property specifies the color of the bar code image borders, while their width is specified by the [BorderWidth](#) property.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

BorderWidth Property

Specifies the border width of the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property BorderWidth As Single
```

C#

```
public Single BorderWidth { get; set; }
```

Property Value

A [System.Single](#) value that is the border width, measured in units specified by the [Unit](#) property.

Remarks

If the **BorderWidth** property value is **0**, bar code borders are invisible.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

BottomCaption Property

Provides access to characteristics of the bar code bottom caption.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property BottomCaption As BarCodeCaption
```

C#

```
public BarCodeCaption BottomCaption { get; }
```

Property Value

A [BarCodeCaption](#) object containing caption characteristics.

Remarks

Use the **BottomCaption** property to specify the caption text, color and font, and the location of the text within a bar code image.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

Code Property

Get the processed code text actually shown in the bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Code As String
```

C#

```
public string Code { get; }
```

Property Value

A string that is the resulting code displayed by the bar code.

Remarks

The [Code](#) and [CodeBinaryData](#) properties allows you to specify the initial data to be coded in the bar code - text or binary data respectively.

The bar code (depending on is [Symbology](#)) may require including a checksum digit or adding trailing (leading) zeros to the initial data for display. The **Code** property gets the resulting code text with the check sum included (if the **CalcChecksum** property is **true**) and padded with zero(s) if required.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

CodeBinaryData Property

Gets or sets the byte array to be coded into certain bar code types.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CodeBinaryData As Byte[]
```

C#

```
public Byte[] CodeBinaryData { get; set; }
```

Property Value

An array of bytes to be coded using the bar code's symbology.

Remarks

Use the **CodeBinaryData** property to code a byte array into a bar code. Only applicable to certain bar code types, such as the [Data Matrix \(ECC200\)](#), [GS1- Data Matrix](#), [PDF417](#) and [QR Code](#) bar codes.

Set the **CompactionMode** property (specifically, [DataMatrixOptions.CompactionMode](#) or the [PDF417Options.CompactionMode](#), or [QRCodeOptions.CompactionMode](#) respectively) to [DataMatrixCompactionMode.Binary](#) to use data specified by the **CodeBinaryData** property.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

[Data Matrix \(ECC200\)](#)

[GS1- Data Matrix](#)

[PDF417](#)

[QR Code](#)

CodeText Property

Gets or sets the text to be coded using the bar code symbology.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CodeText As String
```

C#

```
public string CodeText { get; set; }
```

Property Value

A string containing the text represented by a bar code.

Remarks

For certain bar code types, the byte array can be specified instead of text using the [CodeBinaryData](#) property.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

CodeTextFont Property

Specifies the font used to display the bar code text.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CodeTextFont As Font
```

C#

```
public Font CodeTextFont { get; set; }
```

Property Value

A [System.Drawing.Font](#) object that specifies the font attributes.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

CodeTextHorizontalAlignment Property

Get or sets the horizontal alignment of the bar code text within the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CodeTextHorizontalAlignment As StringAlignment
```

C#

```
public StringAlignment CodeTextHorizontalAlignment { get; set; }
```

Property Value

A [System.Drawing.StringAlignment](#) enumeration value that specifies how the bar code text is aligned within the image.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

CodeTextVerticalAlignment Property

Get or sets the vertical alignment of the bar code text within the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CodeTextVerticalAlignment As StringAlignment
```

C#

```
public StringAlignment CodeTextVerticalAlignment { get; set; }
```

Property Value

A [System.Drawing.StringAlignment](#) enumeration value that specifies how the bar code text is aligned within the image.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

Dpi Property

Gets or sets the dpi value used to render the bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Dpi As Single
```

C#

```
public Single Dpi { get; set; }
```

Property Value

A [System.Single](#) value representing the dpi value used to create the bar code image.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

DpiX Property

Gets or sets the dpi value for the X-coordinate used to render the bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property DpiX As Single
```

C#

```
public Single DpiX { get; set; }
```

Property Value

A [System.Single](#) value representing the dpi value used to create the bar code image.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

DpiY Property

Gets or sets the dpi value for the Y-coordinate used to render the bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property DpiY As Single
```

C#

```
public Single DpiY { get; set; }
```

Property Value

A [System.Single](#) value representing the dpi value used to create the bar code image.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

ForeColor Property

Gets or sets the color of bars in the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ForeColor As Color
```

C#

```
public Color ForeColor { get; set; }
```

Property Value

A [System.Drawing.Color](#) used to paint the bars of the bar code.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

ImageHeight Property

Gets or sets the height of the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property ImageHeight As [Single](#)

C#

```
public Single ImageHeight { get; set; }
```

Property Value

A [System.Single](#) value that is the height of an image in units specified by the [Unit](#) property.

Remarks

If the [AutoSize](#) option is in effect, the **ImageHeight** property value is ignored.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

[Unit](#)

[AutoSize](#)

ImageSize Property

Gets or sets the size of the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ImageSize As SizeF
```

C#

```
public SizeF ImageSize { get; set; }
```

Property Value

A [System.Drawing.SizeF](#) structure that is the size of an image in units specified by the [Unit](#) property.

Remarks

If the [AutoSize](#) option is in effect, the **ImageSize** property value is ignored.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

[Unit](#)

[AutoSize](#)

ImageWidth Property

Gets or sets the width of the resulting image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ImageWidth As Single
```

C#

```
public Single ImageWidth { get; set; }
```

Property Value

A [System.Single](#) value that is the width of an image in units specified by the [Unit](#) property.

Remarks

If the [AutoSize](#) option is in effect, the **ImageWidth** property value is ignored.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

[Unit](#)

[AutoSize](#)

IsPrintingAvailable Property

Indicates whether the bar code can be printed and exported to PDF.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property IsPrintingAvailable As Boolean
```

C#

```
public bool IsPrintingAvailable { get; }
```

Property Value

true, if the control can be printed and exported; otherwise, **false**.

Remarks

The bar code can be printed and exported in various formats if the XtraPrinting Library is available on the end-user machine. To print the bar code, use the [Print](#) method.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

[Print](#)

[ShowPrintPreview](#)

Margins Property

Provides access to margin widths set for a bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Margins As BarCodeMargins
```

C#

```
public BarCodeMargins Margins { get; }
```

Property Value

A [BarCodeMargins](#) object containing margin widths for a bar code image.

Remarks

If the [AutoSize](#) is set to **true**, the **Margins** only affects the printout and PDF export.

Use the [BarCodeMargins.Left](#), [BarCodeMargins.Right](#), [BarCodeMargins.Top](#) and [BarCodeMargins.Bottom](#) properties to define margins for the bar code image.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

Module Property

Gets or sets the width of the narrowest bar or space in the bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Module As Double
```

C#

```
public Double Module { get; set; }
```

Property Value

A [System.Double](#) value which represents the width of the narrowest bar or space.

Remarks

The **Module** property cannot be set to a value less than or equal to **0**. Note that if the **Module** property is set to a very small value so that the width of a bar code is too small, a bar code image may be unreadable by a scanner.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

Options Property

Provides access to options specific for various bar code types.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Options As BarCodeOptions
```

C#

```
public BarCodeOptions Options { get; }
```

Property Value

A [BarCodeOptions](#) object containing type-specific options.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

Paddings Property

Gets the padding settings of a bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Paddings As BarCodePadding
```

C#

```
public BarCodePadding Paddings { get; }
```

Property Value

A [BarCodePadding](#) object that contains padding settings.

Remarks

Paddings in a bar code image are considered to be the amount of space between the inner boundaries of the frame containing the bar code (use the [BorderWidth](#) and the [BorderColor](#) to display the frame borders) and the bar code itself.

If the [AutoSize](#) is set to **true**, the **Paddings** and [Margins](#) settings add white space to the image, increasing its dimensions. If the [AutoSize](#) is set to **false**, the image size is fixed and only the **Paddings** settings are used.

Use the [BarCodePadding.Left](#), [BarCodePadding.Right](#), [BarCodePadding.Top](#) and [BarCodePadding.Bottom](#) properties to define paddings in the bar code image.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

RotationAngle Property

Gets or sets the number of degrees the bar code is rotated around the z-axis.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property RotationAngle As Single
```

C#

```
public Single RotationAngle { get; set; }
```

Property Value

A [System.Single](#) value that is the number of degrees.

Remarks

A positive value indicates clockwise rotation; a negative value indicates counterclockwise rotation.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

ShowText Property

Indicates whether the code text is shown.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property ShowText As Boolean
```

C#

```
public bool ShowText { get; }
```

Property Value

true, if the code text is displayed; otherwise, **false**.

Remarks

To display the code text, set the [BarcodeGeneratorOptions.ShowCodeText](#) property to **true**.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

Symbology Property

Gets or sets the bar code type (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Symbology As Symbology
```

C#

```
public Symbology Symbology { get; set; }
```

Property Value

A [Symbology](#) enumeration specifying the current bar code type.

Remarks

Refer to the [Bar Code Types](#) document for a list of supported symbologies.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

[Symbology](#)

[Bar Code Types](#)

TextRenderingHint Property

Gets or sets the bar code text rendering quality.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property TextRenderingHint As TextRenderingHint
```

C#

```
public TextRenderingHint TextRenderingHint { get; set; }
```

Property Value

A System.Drawing.Text.TextRenderingHint enumeration value specifying the bar code rendering quality. The default is **TextRenderingHint.AntiAlias**.

Remarks

By default, a bar code text is drawn with the Anti-Alias text smoothing settings. You can manually change the text quality rendering using the **TextRenderingHint** property.

See the **TextRenderingHint Enumeration** topic in MSDN to learn more.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

TopCaption Property

Provides access to characteristics of the bar code top caption.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property TopCaption As BarCodeCaption
```

C#

```
public BarCodeCaption TopCaption { get; }
```

Property Value

A [BarCodeCaption](#) object containing caption characteristics.

Remarks

Use the **TopCaption** property to specify the caption text, its color and font, and the location of the text within a bar code image.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

Unit Property

Gets or sets the unit of measurement for bar code settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Unit As GraphicsUnit
```

C#

```
public GraphicsUnit Unit { get; set; }
```

Property Value

A [System.Drawing.GraphicsUnit](#) enumeration value

Remarks

The unit of measurement set by this property is used for bar code settings specified via API, unless stated otherwise.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

BarCode Methods

A non-visual object that can generate a bar code for given data with the specified options.

The following tables list the members exposed by the [BarCode](#) type.

Public Methods

	Name	Description
	Dispose	Resets the bar code image, disposes of the BarCode object and releases all the allocated resources.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	ExportToPdf	Overloaded. Exports the bar code image to the specified file path in PDF format.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Print	Prints the bar code to the default printer.
	PrintDialog	Invokes a dialog that enables you to select a printer and change print settings.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	Save	Overloaded. Saves the bar code image to the stream in a specified format.
	ShowPrintPreview	Invokes a dialog that enables you to preview the printout, print or save it in PDF format, or as an image file.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

Dispose Method

Resets the bar code image, disposes of the BarCode object and releases all the allocated resources.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Dispose()
```

C#

```
public void Dispose()
```

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

ExportToPdf Method

Exports the bar code to the specified stream in PDF format.

Overload List

Name	Description
void ExportToPdf(Stream stream)	Exports the bar code to the specified stream in PDF format.
void ExportToPdf(string fileName)	Exports the bar code image to the specified file path in PDF format.

See Also
[BarCode Class](#)
[BarCode Members](#)
[DevExpress.BarCodes Namespace](#)
[BarCode.ExportToPdf Overload List](#)

Exports the bar code to the specified stream in PDF format.

Namespace: [DevExpress.BarCodes](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToPdf(  
    ByVal stream As Stream  
)
```

C#

```
public void ExportToPdf(  
    Stream stream  
)
```

Parameters

stream
A [System.IO.Stream](#) object to which the created document is exported.

Remarks

To print the bar code, use the [Print](#) or the [ShowPrintPreview](#) methods.

See Also
[BarCode Class](#)
[BarCode Members](#)
[DevExpress.BarCodes Namespace](#)
[BarCode.ExportToPdf Overload List](#)
[Print](#)
[ShowPrintPreview](#)

Exports the bar code image to the specified file path in PDF format.

Namespace: [DevExpress.BarCodes](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToPdf(  
    ByVal fileName As String  
)
```

C#

```
public void ExportToPdf(  
    string fileName  
)
```

Parameters*fileName*

A [System.String](#) which specifies the file name (including the full path) for the created PDF file.

Remarks

To print the bar code, use the [Print](#) or the [ShowPrintPreview](#) methods.

See Also[BarCode Class](#)[BarCode Members](#)[DevExpress.BarCodes Namespace](#)[BarCode.ExportToPdf Overload List](#)[Print](#)[ShowPrintPreview](#)

Print Method

Prints the bar code to the default printer.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Print()
```

C#

```
public void Print()
```

Remarks

Use the [PrintDialog](#) method to invoke a dialog that allows you to select a printer and change printer settings.

The [ShowPrintPreview](#) method enables you to preview the printout.

To print to PDF, use the [ExportToPdf](#) method.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

[PrintDialog](#)

[ShowPrintPreview](#)

[ExportToPdf](#)

PrintDialog Method

Invokes a dialog that enables you to select a printer and change print settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub PrintDialog()
```

C#

```
public void PrintDialog()
```

Remarks

Use the [Print](#) method to print a bar code to the default printer.

The [ShowPrintPreview](#) method enables you to preview the printout.

To print to PDF, use the [ExportToPdf](#) method.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

[Print](#)

[ShowPrintPreview](#)

[ExportToPdf](#)

Save Method

Saves the bar code image to a file, specifying the image file format.

Overload List

Name	Description
void Save(string fileName, ImageFormat format)	Saves the bar code image to a file, specifying the image file format.
void Save(Stream stream, ImageFormat format)	Saves the bar code image to the stream in a specified format.

See Also
[BarCode Class](#)
[BarCode Members](#)
[DevExpress.BarCodes Namespace](#)
[BarCode.Save Overload List](#)

Saves the bar code image to a file, specifying the image file format.

Namespace: [DevExpress.BarCodes](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Save(  
    ByVal fileName As String,  
    ByVal format As ImageFormat  
)
```

C#

```
public void Save(  
    string fileName,  
    ImageFormat format  
)
```

Parameters

fileName
A [System.String](#) value, which specifies the path to the file into which to save the image.
format
A [System.Drawing.Imaging.ImageFormat](#) enumeration that specifies the image format.

See Also
[BarCode Class](#)
[BarCode Members](#)
[DevExpress.BarCodes Namespace](#)
[BarCode.Save Overload List](#)

Saves the bar code image to the stream in a specified format.

Namespace: [DevExpress.BarCodes](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Save(  
    ByVal stream As Stream,  
    ByVal format As ImageFormat  
)
```

C#

```
public void Save(  
    Stream stream,  
    ImageFormat format  
)
```

Parameters

stream

A [System.IO.Stream](#) descendant to which the image is written.

format

A [System.Drawing.Imaging.ImageFormat](#) enumeration that specifies an image format.

See Also

[Barcode Class](#)

[Barcode Members](#)

[DevExpress.BarCodes Namespace](#)

[Barcode.Save Overload List](#)

ShowPrintPreview Method

Invokes a dialog that enables you to preview the printout, print or save it in PDF format, or as an image file.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ShowPrintPreview()
```

C#

```
public void ShowPrintPreview()
```

Remarks

Use the [PrintDialog](#) to invoke a dialog that allows you to change printer settings.

The [Print](#) method enables you to print to the default printer.

To print to PDF, use the [ExportToPdf](#) method.

See Also

[BarCode Class](#)

[BarCode Members](#)

[DevExpress.BarCodes Namespace](#)

[Print](#)

[ShowPrintPreview](#)

[ExportToPdf](#)

BarcodeCaption Class

Provides access to the bar code caption characteristics.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class BarcodeCaption Inherits [Object](#)

C#

public class BarcodeCaption : [object](#)

Remarks

The top bar code caption is accessible via the [Barcode.TopCaption](#) property, and the bottom bar code caption is accessible via the [Barcode.BottomCaption](#) property.

EAN8



EAN-8 is the EAN equivalent of UPC-E in the sense that it provides a "short" barcode for small packages.

Barcode properties

☒ Show text

Text: 01234565

Vert Align: Center

Horz Align: Center

Font Name: Sylfaen

Angle: 45

Fore Color: 0, 128, 0

Back Color: White

Top Caption

☒ Show Caption

Text: top caption

Alignment: Center

Fore Color: 255, 128, 0

Font Name:

Bottom Caption

☒ Show Caption

Text: bottom caption

Alignment: Center

Fore Color: 255, 128, 0

Font Name:

Inheritance Hierarchy

[Object](#)

BarcodeCaption

See Also[BarCodeCaption Members](#)[DevExpress.BarCodes Namespace](#)

BarCodeCaption Members







Provides access to the bar code caption characteristics.

The following tables list the members exposed by the [BarCodeCaption](#) type.

Public Properties

	Name	Description
	Font	Gets or sets the typeface, size and style used to display the bar code caption.
	ForeColor	Gets or sets the color of the text used to display the bar code caption.
	HorizontalAlignment	Gets or sets the horizontal alignment of the text displayed in the bar code caption.
	Paddings	Provides access to paddings that specify the space between the bar code caption and the bar code edge.
	Text	Gets or sets the text of the bar code caption.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BarCodeCaption Members](#)
[DevExpress.BarCodes Namespace](#)

BarCodeCaption Properties







Provides access to the bar code caption characteristics.

The following tables list the members exposed by the [BarCodeCaption](#) type.

Public Properties

	Name	Description
	Font	Gets or sets the typeface, size and style used to display the bar code caption.
	ForeColor	Gets or sets the color of the text used to display the bar code caption.
	HorizontalAlignment	Gets or sets the horizontal alignment of the text displayed in the bar code caption.
	Paddings	Provides access to paddings that specify the space between the bar code caption and the bar code edge.
	Text	Gets or sets the text of the bar code caption.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BarCodeCaption Members](#)
[DevExpress.BarCodes Namespace](#)

Font Property

Gets or sets the typeface, size and style used to display the bar code caption.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Font As Font
```

C#

```
public Font Font { get; set; }
```

Property Value

A [System.Drawing.Font](#) object that is the text font specifying the typeface, size and style of the text.

See Also

[BarCodeCaption Class](#)

[BarCodeCaption Members](#)

[DevExpress.BarCodes Namespace](#)

ForeColor Property

Gets or sets the color of the text used to display the bar code caption.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ForeColor As Color
```

C#

```
public Color ForeColor { get; set; }
```

Property Value

A [System.Drawing.Color](#) object that specifies the text color.

See Also

[BarcodeCaption Class](#)

[BarcodeCaption Members](#)

[DevExpress.BarCodes Namespace](#)

HorizontalAlignment Property

Gets or sets the horizontal alignment of the text displayed in the bar code caption.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property HorizontalAlignment As [StringAlignment](#)

C#

```
public StringAlignment HorizontalAlignment { get; set; }
```

Property Value

A [System.Drawing.StringAlignment](#) enumeration member that specifies the alignment of the text string relative to its layout rectangle.

See Also

[BarcodeCaption Class](#)

[BarcodeCaption Members](#)

[DevExpress.BarCodes Namespace](#)

Paddings Property

Provides access to paddings that specify the space between the bar code caption and the bar code edge.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Paddings As BarCodePadding
```

C#

```
public BarCodePadding Paddings { get; }
```

Property Value

A [BarCodePadding](#) object containing information on paddings associated with a bar code caption.

Remarks

The bar code caption text is specified by the [Text](#) property.

The paddings for the bar code image are specified by using the [BarCode.Paddings](#) property. The margins for the bar code image are specified by using the [BarCode.Margins](#) property.

See Also

[BarCodeCaption Class](#)

[BarCodeCaption Members](#)

[DevExpress.BarCodes Namespace](#)

Text Property

Gets or sets the text of the bar code caption.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Text As String
```

C#

```
public string Text { get; set; }
```

Property Value

A string that is the text displayed in the bar code caption.

Remarks

The text for the top caption can be specified by using **barCode.TopCaption.Text** notation. The text for the bottom caption can be specified by using **barCode.BottomCaption.Text** notation.

The text encoded in the bar code is specified via the [BarCode.CodeText](#) property and displayed by setting the [BarCode.ShowText](#) property to **true**.

See Also

[BarCodeCaption Class](#)

[BarCodeCaption Members](#)

[DevExpress.BarCodes Namespace](#)

[BarCode.CodeText](#)

[BarCode.ShowText](#)

BarcodeGeneratorOptions Class

Base class for options specific to a certain bar code symbology.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class BarcodeGeneratorOptions Inherits [Object](#)

C#

public abstract class BarcodeGeneratorOptions : [object](#)

Inheritance Hierarchy

[Object](#)

BarcodeGeneratorOptions

Derived classes

See Also

[BarcodeGeneratorOptions Members](#)


[DevExpress.BarCodes Namespace](#)

BarcodeGeneratorOptions Members







Base class for options specific to a certain bar code symbology.

The following tables list the members exposed by the [BarcodeGeneratorOptions](#) type.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BarcodeGeneratorOptions Members](#)


[DevExpress.BarCodes Namespace](#)

BarCodeGeneratorOptions Properties







Base class for options specific to a certain bar code symbology.

The following tables list the members exposed by the [BarCodeGeneratorOptions](#) type.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BarCodeGeneratorOptions Members](#)
[DevExpress.BarCodes Namespace](#)

ShowCodeText Property

Gets or sets whether bar code text data should be displayed in a bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ShowCodeText As Boolean
```

C#

```
public bool ShowCodeText { get; set; }
```

Property Value

true, to display text data; otherwise, **false**.

Remarks

Bar code text data is specified by using the [Barcode.CodeText](#) property.

See Also

[BarcodeGeneratorOptions Class](#)

[BarcodeGeneratorOptions Members](#)

[DevExpress.BarCodes Namespace](#)

BarCodeMargins Class

Contains values used to specify bar code margins.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

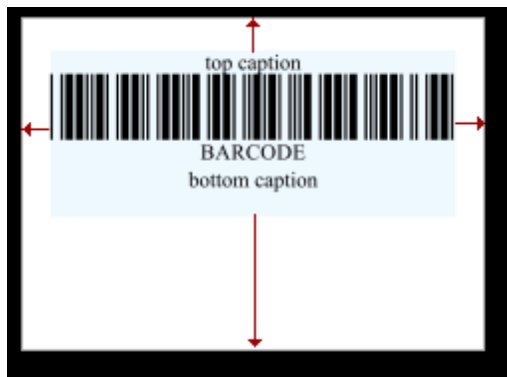
Public Class BarCodeMargins Inherits [Object](#)

C#

public class BarCodeMargins : [object](#)

Remarks

The margins for the bar code image are illustrated in the following picture.



Inheritance Hierarchy

[Object](#)

BarCodeMargins

See Also

[BarCodeMargins Members](#)


[DevExpress.BarCodes Namespace](#)

BarCodeMargins Members





Contains values used to specify bar code margins.

The following tables list the members exposed by the [BarCodeMargins](#) type.







Public Constructors

	Name	Description
	BarCodeMargins	Initializes a new instance of the BarCodeMargins class with default settings.

Public Properties

	Name	Description
	Bottom	Gets or sets the bottom margin for the bar code image.
	Left	Gets or sets the left margin for the bar code image.
	Right	Gets or sets the right margin for the bar code image.
	Top	Gets or sets the top margin for the bar code image.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BarCodeMargins Members](#)

[DevExpress.BarCodes Namespace](#)

BarCodeMargins Constructor

Initializes a new instance of the [BarCodeMargins](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public BarCodeMargins()
```

See Also

[BarCodeMargins Class](#)

[BarCodeMargins Members](#)

[DevExpress.BarCodes Namespace](#)

BarCodeMargins Properties





Contains values used to specify bar code margins.

The following tables list the members exposed by the [BarCodeMargins](#) type.







Public Constructors

	Name	Description
	BarCodeMargins	Initializes a new instance of the BarCodeMargins class with default settings.

Public Properties

	Name	Description
	Bottom	Gets or sets the bottom margin for the bar code image.
	Left	Gets or sets the left margin for the bar code image.
	Right	Gets or sets the right margin for the bar code image.
	Top	Gets or sets the top margin for the bar code image.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BarCodeMargins Members](#)

[DevExpress.BarCodes Namespace](#)

Bottom Property

Gets or sets the bottom margin for the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Bottom As Single
```

C#

```
public Single Bottom { get; set; }
```

Property Value

A [System.Single](#) value that specifies the margin in pixels.

See Also

[BarCodeMargins Class](#)

[BarCodeMargins Members](#)

[DevExpress.BarCodes Namespace](#)

Left Property

Gets or sets the left margin for the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Left As Single
```

C#

```
public Single Left { get; set; }
```

Property Value

A [System.Single](#) value that specifies the margin in pixels.

See Also

[BarCodeMargins Class](#)

[BarCodeMargins Members](#)

[DevExpress.BarCodes Namespace](#)

Right Property

Gets or sets the right margin for the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Right As Single
```

C#

```
public Single Right { get; set; }
```

Property Value

A [System.Single](#) value that specifies the margin in pixels.

See Also

[BarCodeMargins Class](#)

[BarCodeMargins Members](#)

[DevExpress.BarCodes Namespace](#)

Top Property

Gets or sets the top margin for the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Top As Single
```

C#

```
public Single Top { get; set; }
```

Property Value

A [System.Single](#) value that specifies the margin in pixels.

See Also

[BarCodeMargins Class](#)

[BarCodeMargins Members](#)

[DevExpress.BarCodes Namespace](#)

BarCodeOptions Class

Groups options that are specific to a certain symbology.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class BarCodeOptions Inherits Object
```

C#

```
public class BarCodeOptions : object
```

Remarks

Use a corresponding property of the **BarCodeOptions** class to access options specific for a certain symbology (bar code type).

Inheritance Hierarchy

[Object](#)

BarCodeOptions

See Also

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

[Bar Code Types](#)

BarCodeOptions Members













Groups options that are specific to a certain symbology.

The following tables list the members exposed by the [BarCodeOptions](#) type.

Public Constructors




	Name	Description
	BarCodeOptions	Initializes a new instance of the BarCodeOptions class with the default settings.

Public Properties

	Name	Description
	Codabar	Provides access to options specific to this type of bar code (symbology).
	Code11	Provides access to options specific to this type of bar code (symbology).
	Code128	Provides access to options specific to this type of bar code (symbology).
	Code39	Provides access to options specific to this type of bar code (symbology).
	Code39Extended	Provides access to options specific to this type of bar code (symbology).
	Code93	Provides access to options specific to this type of bar code (symbology).
	Code93Extended	Provides access to options specific to this type of bar code (symbology).
	CodeMSI	Provides access to options specific to this type of bar code (symbology).
	DataBar	Provides access to options specific to this type of bar code (symbology).
	DataMatrix	Provides access to options specific to this type of bar code (symbology).
	DataMatrixGS1	Provides access to options specific for this type of bar code (symbology).
	EAN128	Provides access to options specific to this type of bar code (symbology).
	EAN13	Provides access to options specific to this type of bar code (symbology).
	EAN8	Provides access to options specific to this type of bar code (symbology).
	Industrial2of5	Provides access to options specific to this type of bar code (symbology).
	IntelligentMail	Provides access to options specific to this type of bar code (symbology).

	Interleaved2of5	Provides access to options specific to this type of bar code (symbology).
	ITF14	Provides access to options specific to ITF14 type (Interleaved Two of Five for GS1) bar code (symbology).
	Matrix2of5	Provides access to options specific to this type of bar code (symbology).
	PDF417	Provides access to options specific to this type of bar code (symbology).
	PostNet	Provides access to options specific to this type of bar code (symbology).
	QRCode	Provides access to options specific to this type of bar code (symbology).
	UPCA	Provides access to options specific to this type of bar code (symbology).
	UPCE0	Provides access to options specific to this type of bar code (symbology).
	UPCE1	Provides access to options specific to this type of bar code (symbology).
	UPCSupplemental2	Provides access to options specific to this type of bar code (symbology).
	UPCSupplemental5	Provides access to options specific to this type of bar code (symbology).

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[BarCodeOptions Members](#)[DevExpress.BarCodes Namespace](#)[Bar Code Types](#)

BarCodeOptions Constructor

Initializes a new instance of the [BarCodeOptions](#) class with the default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public BarCodeOptions()
```

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

BarCodeOptions Properties













Groups options that are specific to a certain symbology.

The following tables list the members exposed by the [BarCodeOptions](#) type.

Public Constructors







	Name	Description
	BarCodeOptions	Initializes a new instance of the BarCodeOptions class with the default settings.

Public Properties

	Name	Description
	Codabar	Provides access to options specific to this type of bar code (symbology).
	Code11	Provides access to options specific to this type of bar code (symbology).
	Code128	Provides access to options specific to this type of bar code (symbology).
	Code39	Provides access to options specific to this type of bar code (symbology).
	Code39Extended	Provides access to options specific to this type of bar code (symbology).
	Code93	Provides access to options specific to this type of bar code (symbology).
	Code93Extended	Provides access to options specific to this type of bar code (symbology).
	CodeMSI	Provides access to options specific to this type of bar code (symbology).
	DataBar	Provides access to options specific to this type of bar code (symbology).
	DataMatrix	Provides access to options specific to this type of bar code (symbology).
	DataMatrixGS1	Provides access to options specific for this type of bar code (symbology).
	EAN128	Provides access to options specific to this type of bar code (symbology).
	EAN13	Provides access to options specific to this type of bar code (symbology).
	EAN8	Provides access to options specific to this type of bar code (symbology).
	Industrial2of5	Provides access to options specific to this type of bar code (symbology).
	IntelligentMail	Provides access to options specific to this type of bar code (symbology).

	Interleaved2of5	Provides access to options specific to this type of bar code (symbology).
	ITF14	Provides access to options specific to ITF14 type (Interleaved Two of Five for GS1) bar code (symbology).
	Matrix2of5	Provides access to options specific to this type of bar code (symbology).
	PDF417	Provides access to options specific to this type of bar code (symbology).
	PostNet	Provides access to options specific to this type of bar code (symbology).
	QRCode	Provides access to options specific to this type of bar code (symbology).
	UPCA	Provides access to options specific to this type of bar code (symbology).
	UPCE0	Provides access to options specific to this type of bar code (symbology).
	UPCE1	Provides access to options specific to this type of bar code (symbology).
	UPCSupplemental2	Provides access to options specific to this type of bar code (symbology).
	UPCSupplemental5	Provides access to options specific to this type of bar code (symbology).

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[BarCodeOptions Members](#)[DevExpress.BarCodes Namespace](#)[Bar Code Types](#)

Codabar Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Codabar As CodabarOptions
```

C#

```
public CodabarOptions Codabar { get; set; }
```

Property Value

A [CodabarOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Code11 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Code11 As Code11Options
```

C#

```
public Code11Options Code11 { get; set; }
```

Property Value

A [Code11Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Code128 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Code128 As Code128Options
```

C#

```
public Code128Options Code128 { get; set; }
```

Property Value

A [Code128Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Code39 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Code39 As Code39Options
```

C#

```
public Code39Options Code39 { get; set; }
```

Property Value

A [Code39Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Code39Extended Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Code39Extended As Code39ExtendedOptions
```

C#

```
public Code39ExtendedOptions Code39Extended { get; set; }
```

Property Value

A [Code39ExtendedOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Code93 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Code93 As Code93Options
```

C#

```
public Code93Options Code93 { get; set; }
```

Property Value

A [Code93Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Code93Extended Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Code93Extended As Code93ExtendedOptions
```

C#

```
public Code93ExtendedOptions Code93Extended { get; set; }
```

Property Value

A [Code93ExtendedOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

CodeMSI Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CodeMSI As CodeMSIOptions
```

C#

```
public CodeMSIOptions CodeMSI { get; set; }
```

Property Value

A [CodeMSIOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

DataBar Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property DataBar As DataBarOptions
```

C#

```
public DataBarOptions DataBar { get; set; }
```

Property Value

A [DataBarOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

DataMatrix Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property DataMatrix As DataMatrixOptions
```

C#

```
public DataMatrixOptions DataMatrix { get; set; }
```

Property Value

A [DataMatrixOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

DataMatrixGS1 Property

Provides access to options specific for this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property DataMatrixGS1 As DataMatrixGS1Options
```

C#

```
public DataMatrixGS1Options DataMatrixGS1 { get; set; }
```

Property Value

A [DataMatrixGS1Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

EAN128 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property EAN128 As EAN128Options
```

C#

```
public EAN128Options EAN128 { get; set; }
```

Property Value

A [EAN128Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

EAN13 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property EAN13 As [EAN13Options](#)

C#

```
public EAN13Options EAN13 { get; set; }
```

Property Value

A [EAN13Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

EAN8 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property EAN8 As EAN8Options
```

C#

```
public EAN8Options EAN8 { get; set; }
```

Property Value

A [EAN8Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Industrial2of5 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Industrial2of5 As Industrial2of5Options
```

C#

```
public Industrial2of5Options Industrial2of5 { get; set; }
```

Property Value

A [Industrial2of5Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

IntelligentMail Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property IntelligentMail As IntelligentMailOptions
```

C#

```
public IntelligentMailOptions IntelligentMail { get; set; }
```

Property Value

A [IntelligentMailOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Interleaved2of5 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Interleaved2of5 As Interleaved2of5Options
```

C#

```
public Interleaved2of5Options Interleaved2of5 { get; set; }
```

Property Value

A [UPCSupplemental5Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

ITF14 Property

Provides access to options specific to **ITF14** type (**Interleaved Two of Five** for **GS1**) bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ITF14 As Interleaved2of5Options
```

C#

```
public Interleaved2of5Options ITF14 { get; set; }
```

Property Value

An [Interleaved2of5Options](#) class instance containing bar code specific options.

Remarks

ITF-14 (Interleaved Two of Five) is the **GS1** implementation of an **Interleaved 2 of 5** bar code to encode a Global Trade Item Number.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Matrix2of5 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Matrix2of5 As Matrix2of5Options
```

C#

```
public Matrix2of5Options Matrix2of5 { get; set; }
```

Property Value

A [Matrix2of5Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

PDF417 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property PDF417 As PDF417Options
```

C#

```
public PDF417Options PDF417 { get; }
```

Property Value

A [PDF417Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

PostNet Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property PostNet As PostNetOptions
```

C#

```
public PostNetOptions PostNet { get; set; }
```

Property Value

A [UPCSupplemental5Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

QRCode Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property QRCode As QRCodeOptions
```

C#

```
public QRCodeOptions QRCode { get; set; }
```

Property Value

A [QRCodeOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

UPCA Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property UPCA As UPCAOptions
```

C#

```
public UPCAOptions UPCA { get; set; }
```

Property Value

A [UPCAOptions](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

UPCE0 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property UPCE0 As UPCE0Options
```

C#

```
public UPCE0Options UPCE0 { get; set; }
```

Property Value

A [UPCE0Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

UPCE1 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property UPCE1 As UPCE1Options
```

C#

```
public UPCE1Options UPCE1 { get; set; }
```

Property Value

A [UPCE1Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

UPCSupplemental2 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property UPCSupplemental2 As UPCSupplemental2Options
```

C#

```
public UPCSupplemental2Options UPCSupplemental2 { get; set; }
```

Property Value

A [UPCSupplemental2Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

UPCSupplemental5 Property

Provides access to options specific to this type of bar code (symbology).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property UPCSupplemental5 As UPCSupplemental5Options
```

C#

```
public UPCSupplemental5Options UPCSupplemental5 { get; set; }
```

Property Value

A [UPCSupplemental5Options](#) class instance containing bar code specific options.

See Also

[BarCodeOptions Class](#)

[BarCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

BarCodePadding Class

Contains values used to specify bar code paddings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

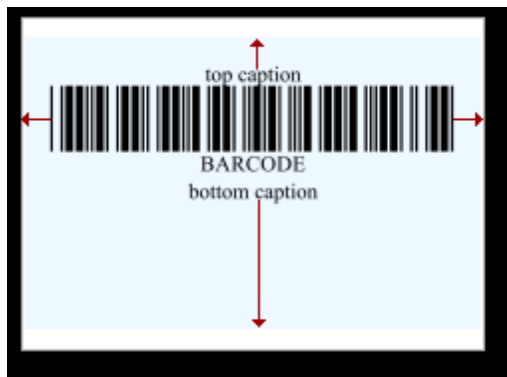
Public Class BarCodePadding Inherits [Object](#)

C#

public class BarCodePadding : [object](#)

Remarks

The paddings for the bar code image are illustrated in the following picture.



Inheritance Hierarchy

[Object](#)

BarCodePadding

See Also

[BarCodePadding Members](#)


[DevExpress.BarCodes Namespace](#)

BarCodePadding Members



Contains values used to specify bar code paddings.

The following tables list the members exposed by the [BarCodePadding](#) type.







Public Constructors

	Name	Description
	BarCodePadding	Initializes a new instance of the BarCodePadding class with default settings.

Public Properties

	Name	Description
	Bottom	Gets or sets the bottom padding for the bar code image.
	Left	Gets or sets the left padding for the bar code image.
	Right	Gets or sets the right padding for the bar code image.
	Top	Gets or sets the top padding for the bar code image.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[BarCodePadding Members](#)

[DevExpress.BarCodes Namespace](#)

BarCodePadding Constructor

Initializes a new instance of the [BarCodePadding](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public BarCodePadding()
```

See Also

[BarCodePadding Class](#)

[BarCodePadding Members](#)

[DevExpress.BarCodes Namespace](#)

BarCodePadding Properties





Contains values used to specify bar code paddings.

The following tables list the members exposed by the [BarCodePadding](#) type.







Public Constructors

	Name	Description
	BarCodePadding	Initializes a new instance of the BarCodePadding class with default settings.

Public Properties

	Name	Description
	Bottom	Gets or sets the bottom padding for the bar code image.
	Left	Gets or sets the left padding for the bar code image.
	Right	Gets or sets the right padding for the bar code image.
	Top	Gets or sets the top padding for the bar code image.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BarCodePadding Members](#)

[DevExpress.BarCodes Namespace](#)

Bottom Property

Gets or sets the bottom padding for the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Bottom As Single
```

C#

```
public Single Bottom { get; set; }
```

Property Value

A [System.Single](#) value that specifies the padding in pixels.

See Also

[BarCodePadding Class](#)

[BarCodePadding Members](#)

[DevExpress.BarCodes Namespace](#)

Left Property

Gets or sets the left padding for the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Left As Single
```

C#

```
public Single Left { get; set; }
```

Property Value

A [System.Single](#) value that specifies the padding in pixels.

See Also

[BarCodePadding Class](#)

[BarCodePadding Members](#)

[DevExpress.BarCodes Namespace](#)

Right Property

Gets or sets the right padding for the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Right As Single
```

C#

```
public Single Right { get; set; }
```

Property Value

A [System.Single](#) value that specifies the padding in pixels.

See Also

[BarCodePadding Class](#)

[BarCodePadding Members](#)

[DevExpress.BarCodes Namespace](#)

Top Property

Gets or sets the top padding for the bar code image.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Top As Single
```

C#

```
public Single Top { get; set; }
```

Property Value

A [System.Single](#) value that specifies the padding in pixels.

See Also

[BarCodePadding Class](#)

[BarCodePadding Members](#)

[DevExpress.BarCodes Namespace](#)

CodabarOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class CodabarOptions Inherits [BarCodeGeneratorOptions](#)

C#

public class CodabarOptions : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

CodabarOptions

See Also

[CodabarOptions Members](#)

[DevExpress.BarCodes Namespace](#)

[Codabar](#)

CodabarOptions Members

Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [CodabarOptions](#) type.







Public Constructors

	Name	Description
	CodabarOptions	Initializes a new instance of the CodabarOptions class with the default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	StartStopPair	Gets or sets the first (start) and last (stop) symbols used to code the bar code's structure.
	WideNarrowRatio	Gets or sets the density of bars.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[CodabarOptions Members](#)
[DevExpress.BarCodes Namespace](#)

[Codabar](#)

CodabarOptions Constructor

Initializes a new instance of the [CodabarOptions](#) class with the default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public CodabarOptions()
```

See Also

[CodabarOptions Class](#)

[CodabarOptions Members](#)

[DevExpress.BarCodes Namespace](#)

CodabarOptions Properties

Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [CodabarOptions](#) type.







Public Constructors

	Name	Description
	CodabarOptions	Initializes a new instance of the CodabarOptions class with the default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	StartStopPair	Gets or sets the first (start) and last (stop) symbols used to code the bar code's structure.
	WideNarrowRatio	Gets or sets the density of bars.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[CodabarOptions Members](#)
[DevExpress.BarCodes Namespace](#)

[Codabar](#)

StartStopPair Property

Gets or sets the first (start) and last (stop) symbols used to code the bar code's structure.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property StartStopPair As CodabarStartStopPair
```

C#

```
public CodabarStartStopPair StartStopPair { get; set; }
```

Property Value

A [CodabarStartStopPair](#) enumeration value that specifies the pair of start and stop symbols used. The default value is [CodabarStartStopPair.AT](#).

Remarks

Note that the start and stop symbol bars are added to the bar code automatically, so you don't need to specify them in the bar code's `DevExpress.XtraReports.UI.XRControl.Text` property.

See Also

[CodabarOptions Class](#)

[CodabarOptions Members](#)

[DevExpress.BarCodes Namespace](#)

WideNarrowRatio Property

Gets or sets the density of bars.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property WideNarrowRatio As Single
```

C#

```
public Single WideNarrowRatio { get; set; }
```

Property Value

A float value specifying the density of bars. The default is **2**.

Remarks

Use this property to specify the ratio of the thickest bar to the narrowest. The value assigned should be equal to or between **2** and **3**.

See Also

[CodabarOptions Class](#)

[CodabarOptions Members](#)

[DevExpress.BarCodes Namespace](#)

CodabarStartStopPair Enumeration

Lists start/stop patterns used in bar codes.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum CodabarStartStopPair
```

C#

```
public enum CodabarStartStopPair
```

Members

Name	Description
AT	The first element in a bar code is "A" and the last element is "T".
BN	The first element in a bar code is "B" and the last element is "N".
CStar	The first element in a bar code is "C" and the last element is "*".
DE	The first element in a bar code is "D" and the last element is "E".
None	There are no start (first) and stop (last) elements used in a bar code.

See Also

[DevExpress.BarCodes Namespace](#)

Code11Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class Code11Options Inherits [BarCodeGeneratorOptions](#)

C#

public class Code11Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

Code11Options

See Also

[Code11Options Members](#)

[DevExpress.BarCodes Namespace](#)


[Code 11 \(USD-8\)](#)

Code11Options Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code11Options](#) type.







Public Constructors

	Name	Description
	Code11Options	Initializes a new instance of the Code11Options class with the default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Code11Options Members](#)
[DevExpress.BarCodes Namespace](#)
[Code 11 \(USD-8\)](#)

Code11Options Constructor

Initializes a new instance of the [Code11Options](#) class with the default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Code11Options()
```

See Also

[Code11Options Class](#)

[Code11Options Members](#)

[DevExpress.BarCodes Namespace](#)

Code128CharacterSet Enumeration

Lists character sets used in Code128 bar code symbology.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum Code128CharacterSet
```

C#

```
public enum Code128CharacterSet
```

Members

Name	Description
CharsetA	The "A" character set will be used for coding a bar code. It contains ASCII characters 00-95 .
CharsetAuto	The character set will be chosen automatically according to the text assigned to a bar code.
CharsetB	The "B" character set will be used for coding a bar code. It contains ASCII characters 32-127 .
CharsetC	The "C" character set will be used for coding a bar code. It contains numeric digit pairs from 00 to 99 . Each digit pair is coded with one code element, so you can provide 01 23 pairs for coding, but not 123 .

See Also

[DevExpress.BarCodes Namespace](#)

Code128Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class Code128Options Inherits [BarCodeGeneratorOptions](#)

C#

public class Code128Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

Code128Options

See Also

[Code128Options Members](#)

[DevExpress.BarCodes Namespace](#)

[Code 128](#)

Code128Options Members



Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code128Options](#) type.







Public Constructors

	Name	Description
	Code128Options	Initializes a new instance of the Code128Options class with the default settings.

Public Properties

	Name	Description
	Charset	Gets or sets the charset type for the bar code.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Code128Options Members](#)
[DevExpress.BarCodes Namespace](#)
[Code 128](#)

Code128Options Constructor

Initializes a new instance of the [Code128Options](#) class with the default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Code128Options()
```

See Also

[Code128Options Class](#)

[Code128Options Members](#)

[DevExpress.BarCodes Namespace](#)

Code128Options Properties



Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code128Options](#) type.







Public Constructors

	Name	Description
	Code128Options	Initializes a new instance of the Code128Options class with the default settings.

Public Properties

	Name	Description
	Charset	Gets or sets the charset type for the bar code.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Code128Options Members](#)
[DevExpress.BarCodes Namespace](#)
[Code 128](#)

Charset Property

Gets or sets the charset type for the bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property Charset As [Code128CharacterSet](#)

C#

```
public Code128CharacterSet Charset { get; set; }
```

Property Value

A [Code128CharacterSet](#) enumeration value that specifies a charset to be used.

Remarks

Use the **Charset** property to specify the set of symbols which can be used when setting the [Barcode.CodeText](#) property of a [Code 128](#) bar code. If the characters specified in the **CodeText** property are not allowed by the charset, a [System.Exception](#) exception is thrown with the message "There are invalid characters in the text".

Note

If the code text may contain any ASCII symbol, set the **Charset** property value to [Code128CharacterSet.CharsetAuto](#).

See Also

[Code128Options Class](#)

[Code128Options Members](#)

[DevExpress.BarCodes Namespace](#)

Code39ExtendedOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class Code39ExtendedOptions Inherits Code39Options
```

C#

```
public class Code39ExtendedOptions : Code39Options
```

Inheritance Hierarchy

[Object](#)

[BarcodeGeneratorOptions](#)

[Code39Options](#)

Code39ExtendedOptions

See Also

[Code39ExtendedOptions Members](#)

[DevExpress.BarCodes Namespace](#)


[Code 39 Extended](#)

Code39ExtendedOptions Members




Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code39ExtendedOptions](#) type.







Public Constructors

	Name	Description
	Code39ExtendedOptions	Initializes a new instance of the Code39ExtendedOptions class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data. (Inherited from Code39Options)
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	WideNarrowRatio	Gets or sets the wide bar to narrow bar ratio. (Inherited from Code39Options)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[Code39ExtendedOptions Members](#)[DevExpress.BarCodes Namespace](#)[Code 39 Extended](#)

Code39ExtendedOptions Constructor

Initializes a new instance of the [Code39ExtendedOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Code39ExtendedOptions()
```

See Also

[Code39ExtendedOptions Class](#)

[Code39ExtendedOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Code39Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class Code39Options Inherits [BarCodeGeneratorOptions](#)

C#

public class Code39Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

Code39Options

 Derived classes

See Also

[Code39Options Members](#)

[DevExpress.BarCodes Namespace](#)


[Code 39 \(USD-3\)](#)

Code39Options Members




Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code39Options](#) type.







Public Constructors

	Name	Description
	Code39Options	Initializes a new instance of the Code39Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarcodeGeneratorOptions)
	WideNarrowRatio	Gets or sets the wide bar to narrow bar ratio.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Code39Options Members](#)

[DevExpress.BarCodes Namespace](#)
[Code 39 \(USD-3\)](#)

Code39Options Constructor

Initializes a new instance of the [Code39Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Code39Options()
```

See Also

[Code39Options Class](#)

[Code39Options Members](#)

[DevExpress.BarCodes Namespace](#)

Code39Options Properties




Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code39Options](#) type.







Public Constructors

	Name	Description
	Code39Options	Initializes a new instance of the Code39Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarcodeGeneratorOptions)
	WideNarrowRatio	Gets or sets the wide bar to narrow bar ratio.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Code39Options Members](#)

[DevExpress.BarCodes Namespace](#)
[Code 39 \(USD-3\)](#)

CalcChecksum Property

Gets or sets whether a check character should be calculated and added to the bar code data.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CalcChecksum As Boolean
```

C#

```
public bool CalcChecksum { get; set; }
```

Property Value

true, to add a check character; otherwise, **false**.

See Also

[Code39Options Class](#)

[Code39Options Members](#)

[DevExpress.BarCodes Namespace](#)

WideNarrowRatio Property

Gets or sets the wide bar to narrow bar ratio.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property WideNarrowRatio As Single
```

C#

```
public Single WideNarrowRatio { get; set; }
```

Property Value

A [System.Single](#) value that specifies the density of bar code bars. The default is **3**.

Remarks

Use this property to specify the ratio of the thickest bar to the narrowest. The value assigned should be equal to or between **2.2** and **3**.

See Also

[Code39Options Class](#)

[Code39Options Members](#)

[DevExpress.BarCodes Namespace](#)

Code93ExtendedOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class Code93ExtendedOptions Inherits [Code93Options](#)

C#

public class Code93ExtendedOptions : [Code93Options](#)

Inheritance Hierarchy

[Object](#)

[BarcodeGeneratorOptions](#)

[Code93Options](#)

Code93ExtendedOptions

See Also

[Code93ExtendedOptions Members](#)

[DevExpress.BarCodes Namespace](#)


[Code 93 Extended](#)

Code93ExtendedOptions Members



Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code93ExtendedOptions](#) type.







Public Constructors

	Name	Description
	Code93ExtendedOptions	Initializes a new instance of the Code93ExtendedOptions class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data. (Inherited from Code93Options)
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Code93ExtendedOptions Members](#)
[DevExpress.BarCodes Namespace](#)
[Code 93 Extended](#)

Code93ExtendedOptions Constructor

Initializes a new instance of the [Code93ExtendedOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Code93ExtendedOptions()
```

See Also

[Code93ExtendedOptions Class](#)

[Code93ExtendedOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Code93Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class Code93Options Inherits [BarCodeGeneratorOptions](#)

C#

public class Code93Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

Code93Options

Derived classes

See Also

[Code93Options Members](#)

[DevExpress.BarCodes Namespace](#)


[Code 93](#)

Code93Options Members



Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code93Options](#) type.







Public Constructors

	Name	Description
	Code93Options	Initializes a new instance of the Code93Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarcodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Code93Options Members](#)

[DevExpress.BarCodes Namespace](#)

[Code 93](#)

Code93Options Constructor

Initializes a new instance of the [Code93Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Code93Options()
```

See Also

[Code93Options Class](#)

[Code93Options Members](#)

[DevExpress.BarCodes Namespace](#)

Code93Options Properties



Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Code93Options](#) type.







Public Constructors

	Name	Description
	Code93Options	Initializes a new instance of the Code93Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarcodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Code93Options Members](#)

[DevExpress.BarCodes Namespace](#)

[Code 93](#)

CalcChecksum Property

Gets or sets whether a check character should be calculated and added to the bar code data.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CalcChecksum As Boolean
```

C#

```
public bool CalcChecksum { get; set; }
```

Property Value

true, to add a check character; otherwise, **false**.

See Also

[Code93Options Class](#)

[Code93Options Members](#)

[DevExpress.BarCodes Namespace](#)

CodeMSIOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class CodeMSIOptions Inherits BarCodeGeneratorOptions
```

C#

```
public class CodeMSIOptions : BarCodeGeneratorOptions
```

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

CodeMSIOptions

See Also

[CodeMSIOptions Members](#)

[DevExpress.BarCodes Namespace](#)

[MSI - Plessey](#)

CodeMSIOptions Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [CodeMSIOptions](#) type.







Public Constructors

	Name	Description
	CodeMSIOptions	Initializes a new instance of the CodeMSIOptions class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[CodeMSIOptions Members](#)
[DevExpress.BarCodes Namespace](#)
[MSI - Plessey](#)

CodeMSIOptions Constructor

Initializes a new instance of the [CodeMSIOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public CodeMSIOptions()
```

See Also

[CodeMSIOptions Class](#)

[CodeMSIOptions Members](#)

[DevExpress.BarCodes Namespace](#)

DataBarOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class DataBarOptions Inherits [BarCodeGeneratorOptions](#)

C#

public class DataBarOptions : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

DataBarOptions

See Also

[DataBarOptions Members](#)


[DevExpress.BarCodes Namespace](#)

DataBarOptions Members





Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [DataBarOptions](#) type.


Public Constructors

	Name	Description
	DataBarOptions	Initializes a new instance of the DataBarOptions class with default settings.

Public Properties

	Name	Description
	FNC1Substitute	Specifies the symbol (or set of symbols) in the bar code's text that will be replaced with the FNC1 functional character when the bars are drawn.
	SegmentsInRow	Gets or sets the number of data segments per row in the Expanded Stacked type of a GS1 DataBar bar code.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	Type	Gets or sets the symbol type of the GS1 DataBar family of bar codes.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)

	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
--	--------------------------	--

See Also
[DataBarOptions Members](#)
[DevExpress.BarCodes Namespace](#)

DataBarOptions Constructor

Initializes a new instance of the [DataBarOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public DataBarOptions()
```

See Also

[DataBarOptions Class](#)

[DataBarOptions Members](#)

[DevExpress.BarCodes Namespace](#)

DataBarOptions Properties





Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [DataBarOptions](#) type.






Public Constructors

	Name	Description
	DataBarOptions	Initializes a new instance of the DataBarOptions class with default settings.

Public Properties

	Name	Description
	FNC1Substitute	Specifies the symbol (or set of symbols) in the bar code's text that will be replaced with the FNC1 functional character when the bars are drawn.
	SegmentsInRow	Gets or sets the number of data segments per row in the Expanded Stacked type of a GS1 DataBar bar code.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	Type	Gets or sets the symbol type of the GS1 DataBar family of bar codes.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)

	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
--	--------------------------	--

See Also
[DataBarOptions Members](#)
[DevExpress.BarCodes Namespace](#)

FNC1Substitute Property

Specifies the symbol (or set of symbols) in the bar code's text that will be replaced with the **FNC1** functional character when the bars are drawn.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property FNC1Substitute As String
```

C#

```
public string FNC1Substitute { get; set; }
```

Property Value

A [System.String](#) value, specifying the symbols to be replaced.

Remarks

The string, specified by the **FNC1Substitute** property, is replaced in the string assigned to the [Barcode.CodeText](#) property with the invisible **FNC1** character. The **FNC1** character serves as a field separator in the bar code. Thus, the **FNC1Substitute** string is not displayed in the bar code text.

See Also

[DataBarOptions Class](#)

[DataBarOptions Members](#)

[DevExpress.BarCodes Namespace](#)

SegmentsInRow Property

Gets or sets the number of data segments per row in the Expanded Stacked type of a GS1 DataBar bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property SegmentsInRow As Integer
```

C#

```
public int SegmentsInRow { get; set; }
```

Property Value

An integer value specifying the number of data segments per row.

See Also

[DataBarOptions Class](#)

[DataBarOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Type Property

Gets or sets the symbol type of the GS1 DataBar family of bar codes.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Type As DataBarType
```

C#

```
public DataBarType Type { get; set; }
```

Property Value

A [DataBarType](#) enumeration member specifying the symbol type.

See Also

[DataBarOptions Class](#)

[DataBarOptions Members](#)

[DevExpress.BarCodes Namespace](#)

DataBarType Enumeration

Lists symbol types of the GS1 DataBar family.

Namespace: [DevExpress.BarCodes](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum DataBarType`
C#
`public enum DataBarType`

Members

Name	Description
Expanded	<div><div>GS1 Databar Expanded</div><div><p>(01)1234567890123-ABCabc</p></div></div>
ExpandedStacked	<div><div>GS1 Databar Expanded Stacked</div><div><p>(01)1234567890123-ABCabc</p></div></div>
Limited	<div><div>GS1 Databar Limited</div><div><p>(01)01234567890128</p></div></div>
Omnidirectional	<div><div>GS1 Databar Omnidirectional</div></div>

	
Stacked	<p>GS1 Databar Stacked</p> 
StackedOmnidirectional	<p>GS1 Databar Stacked Omnidirectional</p> 
Truncated	<p>GS1 Databar Truncated</p> 

See Also
[DevExpress.BarCodes Namespace](#)

DataMatrixCompactionMode Enumeration

Lists compaction modes used in DataMatrix bar code symbology.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum DataMatrixCompactionMode
```

C#

```
public enum DataMatrixCompactionMode
```

Members

Name	Description
ASCII	Encoding mode used to store double digit numerics, ASCII values 0 - 127 , and Extended ASCII values 128 - 255 .
Binary	Encoding mode used to store binary data. They are encoded using 8 bits per symbol.
C40	Encoding mode for upper-case alphanumeric, lower case and special characters.
Edifact	Encoding mode for ASCII values from 32 to 94 .
Text	Encoding mode for lower-case alphanumeric, upper case and special characters.
X12	Encoding mode for ANSI X12 EDI data set.

See Also

[DevExpress.BarCodes Namespace](#)

DataMatrixGS1Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class DataMatrixGS1Options Inherits [DataMatrixOptions](#)

C#

public class DataMatrixGS1Options : [DataMatrixOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

[DataMatrixOptions](#)

DataMatrixGS1Options

See Also

[DataMatrixGS1Options Members](#)

[DevExpress.BarCodes Namespace](#)


[GS1- Data Matrix](#)

DataMatrixGS1Options Members






Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [DataMatrixGS1Options](#) type.



Public Constructors



	Name	Description
	DataMatrixGS1Options	Initializes a new instance of the DataMatrixGS1Options class with default settings.

Public Properties

	Name	Description
	CompactionMode	Gets or sets the encoding scheme used to encode data in the DataMatrix Code. (Inherited from DataMatrixOptions)
	FNC1Substitut	Obsolete. Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the FNC1 functional character.
	FNC1Substitute	Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the FNC1 functional character.
	MatrixSize	Gets or sets the bar code matrix size. (Inherited from DataMatrixOptions)
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)

	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[DataMatrixGS1Options Members](#)
[DevExpress.BarCodes Namespace](#)
[GS1- Data Matrix](#)

DataMatrixGS1Options Constructor

Initializes a new instance of the [DataMatrixGS1Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public DataMatrixGS1Options()
```

See Also

[DataMatrixGS1Options Class](#)

[DataMatrixGS1Options Members](#)

[DevExpress.BarCodes Namespace](#)

DataMatrixGS1Options Properties






Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [DataMatrixGS1Options](#) type.




Public Constructors



	Name	Description
	DataMatrixGS1Options	Initializes a new instance of the DataMatrixGS1Options class with default settings.

Public Properties

	Name	Description
	CompactionMode	Gets or sets the encoding scheme used to encode data in the DataMatrix Code. (Inherited from DataMatrixOptions)
	FNC1Substitut	Obsolete. Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the FNC1 functional character.
	FNC1Substitute	Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the FNC1 functional character.
	MatrixSize	Gets or sets the bar code matrix size. (Inherited from DataMatrixOptions)
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)

	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[DataMatrixGS1Options Members](#)
[DevExpress.BarCodes Namespace](#)
[GS1- Data Matrix](#)

FNC1Substitut Property

NOTE: This member is now obsolete.

This property has bocome obsolete. Use FNC1Substitute property instead.

Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the **FNC1** functional character.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property FNC1Substitut As String
```


C#

```
public string FNC1Substitut { get; set; }
```

Property Value

A [System.String](#) specifying the symbols to be replaced.

Remarks

 Note	
FNC1 for the GS1- Data Matrix bar code has a special meaning and should be the first in the data sequence.	

See Also

[DataMatrixGS1Options Class](#)

[DataMatrixGS1Options Members](#)

[DevExpress.BarCodes Namespace](#)

Bar Code Recognition Specifics

FNC1Substitute Property

Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the **FNC1** functional character.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property FNC1Substitute As String
```

C#

```
public string FNC1Substitute { get; set; }
```

Property Value

A [System.String](#) specifying the symbols to be replaced.

Remarks

 Note	
---	--

FNC1 for the GS1- Data Matrix bar code has a special meaning and should be the first in the data sequence.

See Also

[DataMatrixGS1Options Class](#)

[DataMatrixGS1Options Members](#)

[DevExpress.BarCodes Namespace](#)

DataMatrixOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class DataMatrixOptions Inherits [BarCodeGeneratorOptions](#)

C#

```
public class DataMatrixOptions : BarCodeGeneratorOptions
```

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

DataMatrixOptions

Derived classes

See Also

[DataMatrixOptions Members](#)

[DevExpress.BarCodes Namespace](#)

[Data Matrix \(ECC200\)](#)

DataMatrixOptions Members




Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [DataMatrixOptions](#) type.







Public Constructors

	Name	Description
	DataMatrixOptions	Initializes a new instance of the DataMatrixOptions class with default settings.

Public Properties

	Name	Description
	CompactionMode	Gets or sets the encoding scheme used to encode data in the DataMatrix Code.
	MatrixSize	Gets or sets the bar code matrix size.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[DataMatrixOptions Members](#)
[DevExpress.BarCodes Namespace](#)

[Data Matrix \(ECC200\)](#)

DataMatrixOptions Constructor

Initializes a new instance of the [DataMatrixOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public DataMatrixOptions()
```

See Also

[DataMatrixOptions Class](#)

[DataMatrixOptions Members](#)

[DevExpress.BarCodes Namespace](#)

DataMatrixOptions Properties




Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [DataMatrixOptions](#) type.







Public Constructors

	Name	Description
	DataMatrixOptions	Initializes a new instance of the DataMatrixOptions class with default settings.

Public Properties

	Name	Description
	CompactionMode	Gets or sets the encoding scheme used to encode data in the DataMatrix Code.
	MatrixSize	Gets or sets the bar code matrix size.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[DataMatrixOptions Members](#)
[DevExpress.BarCodes Namespace](#)

[Data Matrix \(ECC200\)](#)

CompactionMode Property

Gets or sets the encoding scheme used to encode data in the DataMatrix Code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CompactionMode As DataMatrixCompactionMode
```

C#

```
public DataMatrixCompactionMode CompactionMode { get; set; }
```

Property Value

A [DataMatrixCompactionMode](#) enumeration member that specifies the encoding scheme.

Remarks

If the **CompactionMode** is set to **Binary**, the data is obtained from the [Barcode.CodeBinaryData](#) property. Otherwise, the bar code data is obtained from the [Barcode.CodeText](#) property.

See Also

[DataMatrixOptions Class](#)

[DataMatrixOptions Members](#)

[DevExpress.BarCodes Namespace](#)

MatrixSize Property

Gets or sets the bar code matrix size.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property MatrixSize As DataMatrixSize
```

C#

```
public DataMatrixSize MatrixSize { get; set; }
```

Property Value

A [DataMatrixSize](#) enumeration value.

See Also

[DataMatrixOptions Class](#)

[DataMatrixOptions Members](#)

[DevExpress.BarCodes Namespace](#)

DataMatrixSize Enumeration

Lists the available data matrix size options related to the Data Matrix (ECC200) bar code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum DataMatrixSize
```

C#

```
public enum DataMatrixSize
```

Members

Name	Description
Matrix104x104	The bar code matrix size is 104x104 .
Matrix10x10	The bar code matrix size is 10x10 .
Matrix120x120	The bar code matrix size is 120x120 .
Matrix12x12	The bar code matrix size is 12x12 .
Matrix12x26	The bar code matrix size is 12x26 .
Matrix12x36	The bar code matrix size is 12x36 .
Matrix132x132	The bar code matrix size is 132x132 .
Matrix144x144	The bar code matrix size is 144x144 .
Matrix14x14	The bar code matrix size is 14x14 .
Matrix16x16	The bar code matrix size is 16x16 .
Matrix16x36	The bar code matrix size is 16x36 .
Matrix16x48	The bar code matrix size is 16x48 .
Matrix18x18	The bar code matrix size is 18x18 .
Matrix20x20	The bar code matrix size is 20x20 .
Matrix22x22	The bar code matrix size is 22x22 .
Matrix24x24	The bar code matrix size is 24x24 .
Matrix26x26	The bar code matrix size is 26x26 .
Matrix32x32	The bar code matrix size is 32x32 .
Matrix36x36	The bar code matrix size is 36x36 .
Matrix40x40	The bar code matrix size is 40x40 .
Matrix44x44	The bar code matrix size is 44x44 .
Matrix48x48	The bar code matrix size is 48x48 .
Matrix52x52	The bar code matrix size is 52x52 .
Matrix64x64	The bar code matrix size is 64x64 .
Matrix72x72	The bar code matrix size is 72x72 .

Matrix80x80	The bar code matrix size is 80x80 .
Matrix88x88	The bar code matrix size is 88x88 .
Matrix8x18	The bar code matrix size is 8x18 .
Matrix8x32	The bar code matrix size is 8x32 .
Matrix96x96	The bar code matrix size is 96x96 .
MatrixAuto	The bar code matrix size is auto-adjusted, depending on the quantity of encoded data.

See Also

[DevExpress.BarCodes Namespace](#)
[Data Matrix \(ECC200\)](#)

EAN128Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class EAN128Options Inherits [BarCodeGeneratorOptions](#)

C#

public class EAN128Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

EAN128Options

See Also

[EAN128Options Members](#)

[DevExpress.BarCodes Namespace](#)

[EAN-128 \(UCC\)](#)

EAN128Options Members



Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [EAN128Options](#) type.




Public Constructors

	Name	Description
	EAN128Options	Initializes a new instance of the EAN128Options class with default settings.

Public Properties

	Name	Description
	Charset	Gets or sets the character set used for encoding.
	FNC1Substitut	Obsolete. Gets or sets the symbol (or set of symbols) that will be replaced in the barcode data with the FNC1 functional character.
	FNC1Substitute	Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the FNC1 functional character.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)

	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
--	--------------------------	--

See Also
[EAN128Options.Members](#)
[DevExpress.BarCodes.Namespace](#)
[EAN-128 \(UCC\)](#)

EAN128Options Constructor

Initializes a new instance of the [EAN128Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public EAN128Options()
```

See Also

[EAN128Options Class](#)

[EAN128Options Members](#)

[DevExpress.BarCodes Namespace](#)

EAN128Options Properties





Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [EAN128Options](#) type.






Public Constructors

	Name	Description
	EAN128Options	Initializes a new instance of the EAN128Options class with default settings.

Public Properties

	Name	Description
	Charset	Gets or sets the character set used for encoding.
	FNC1Substitut	Obsolete. Gets or sets the symbol (or set of symbols) that will be replaced in the barcode data with the FNC1 functional character.
	FNC1Substitute	Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the FNC1 functional character.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)

	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
--	--------------------------	--

See Also

- [EAN128Options.Members](#)
- [DevExpress.BarCodes.Namespace](#)
- [EAN-128 \(UCC\)](#)

Charset Property

Gets or sets the character set used for encoding.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Charset As Code128CharacterSet
```

C#

```
public Code128CharacterSet Charset { get; set; }
```

Property Value

A [Code128CharacterSet](#) enumeration value that specifies a character set.

Remarks

Use the **Charset** property to specify the set of symbols which can be used when setting the [Barcode.CodeText](#) property. If the characters specified for the bar code data are not allowed by the charset, the exception with the message "There are invalid characters in the text" is thrown.

See Also

[EAN128Options Class](#)

[EAN128Options Members](#)

[DevExpress.BarCodes Namespace](#)

FNC1Substitut Property

NOTE: This member is now obsolete.

This property has become obsolete. Use FNC1Substitute property instead.

Gets or sets the symbol (or set of symbols) that will be replaced in the barcode data with the **FNC1** functional character.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property FNC1Substitut As String
```

C#

```
public string FNC1Substitut { get; set; }
```

Property Value

A [System.String](#) specifying the symbols to be replaced.

Remarks

The human-readable text of the EAN128 Code will not contain symbols specified by the **FNC1Substitut** property. They will be used to generate a Function Code One Character (**FNC1**).

See Also

[EAN128Options Class](#)

[EAN128Options Members](#)

[DevExpress.BarCodes Namespace](#)

FNC1Substitute Property

Gets or sets the symbol (or set of symbols) that will be replaced in the bar code data with the **FNC1** functional character.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property FNC1Substitute As String
```

C#

```
public string FNC1Substitute { get; set; }
```

Property Value

A [System.String](#) specifying the symbols to be replaced.

Remarks

The human-readable text of the EAN128 Code will not contain symbols specified by the **FNC1Substitute** property. They will be used to generate a Function Code One Character (**FNC1**).

See Also

[EAN128Options Class](#)

[EAN128Options Members](#)

[DevExpress.BarCodes Namespace](#)

EAN13Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class EAN13Options Inherits [BarcodeGeneratorOptions](#)

C#

public class EAN13Options : [BarcodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarcodeGeneratorOptions](#)

EAN13Options

Derived classes

See Also

[EAN13Options Members](#)

[DevExpress.BarCodes Namespace](#)

[EAN 13](#)

EAN13Options Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [EAN13Options](#) type.







Public Constructors

	Name	Description
	EAN13Options	Initializes a new instance of the EAN13Options class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[EAN13Options Members](#)
[DevExpress.BarCodes Namespace](#)
[EAN 13](#)

EAN13Options Constructor

Initializes a new instance of the [EAN13Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public EAN13Options()
```

See Also

[EAN13Options Class](#)

[EAN13Options Members](#)

[DevExpress.BarCodes Namespace](#)

EAN8Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class EAN8Options Inherits [BarCodeGeneratorOptions](#)

C#

public class EAN8Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

EAN8Options

See Also

[EAN8Options Members](#)

[DevExpress.BarCodes Namespace](#)


[EAN 8](#)

EAN8Options Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [EAN8Options](#) type.







Public Constructors

	Name	Description
	EAN8Options	Initializes a new instance of the EAN8Options class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[EAN8Options Members](#)
[DevExpress.BarCodes Namespace](#)
[EAN 8](#)

EAN8Options Constructor

Initializes a new instance of the [EAN8Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public EAN8Options()
```

See Also

[EAN8Options Class](#)

[EAN8Options Members](#)

[DevExpress.BarCodes Namespace](#)

Industrial2of5Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class Industrial2of5Options Inherits [BarCodeGeneratorOptions](#)

C#

public class Industrial2of5Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

Industrial2of5Options

Derived classes

See Also

[Industrial2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)


[Industrial 2 of 5](#)

Industrial2of5Options Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Industrial2of5Options](#) type.







Public Constructors

	Name	Description
	Industrial2of5Options	Initializes a new instance of the Industrial2of5Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarcodeGeneratorOptions)
	WideNarrowRatio	Gets or sets the wide bar to narrow bar ratio.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Industrial2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)
[Industrial 2 of 5](#)

Industrial2of5Options Constructor

Initializes a new instance of the [Industrial2of5Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Industrial2of5Options()
```

See Also

[Industrial2of5Options Class](#)

[Industrial2of5Options Members](#)


[DevExpress.BarCodes Namespace](#)

Industrial2of5Options Properties


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Industrial2of5Options](#) type.







Public Constructors

	Name	Description
	Industrial2of5Options	Initializes a new instance of the Industrial2of5Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarcodeGeneratorOptions)
	WideNarrowRatio	Gets or sets the wide bar to narrow bar ratio.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Industrial2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)
[Industrial 2 of 5](#)

CalcChecksum Property

Gets or sets whether a check character should be calculated and added to the bar code data.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CalcChecksum As Boolean
```

C#

```
public bool CalcChecksum { get; set; }
```

Property Value

true, to add a check character; otherwise, **false**.

See Also

[Industrial2of5Options Class](#)

[Industrial2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)

WideNarrowRatio Property

Gets or sets the wide bar to narrow bar ratio.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property WideNarrowRatio As Single
```

C#

```
public Single WideNarrowRatio { get; set; }
```

Property Value

A [System.Single](#) value that specifies the density of bar code bars. The default is **2.5**.

Remarks

Use this property to specify the ratio of the thickest bar to the narrowest. The value assigned should be equal to or greater than **2.5**.

See Also

[Industrial2of5Options Class](#)

[Industrial2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)

IntelligentMailOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class IntelligentMailOptions Inherits [BarCodeGeneratorOptions](#)

C#

public class IntelligentMailOptions : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

IntelligentMailOptions

See Also

[IntelligentMailOptions Members](#)

[DevExpress.BarCodes Namespace](#)

[Intelligent Mail](#)

IntelligentMailOptions Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [IntelligentMailOptions](#) type.







Public Constructors

	Name	Description
	IntelligentMailOptions	Initializes a new instance of the IntelligentMailOptions class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[IntelligentMailOptions Members](#)
[DevExpress.BarCodes Namespace](#)
[Intelligent Mail](#)

IntelligentMailOptions Constructor

Initializes a new instance of the [IntelligentMailOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public IntelligentMailOptions()
```

See Also

[IntelligentMailOptions Class](#)

[IntelligentMailOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Interleaved2of5Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class Interleaved2of5Options Inherits [BarCodeGeneratorOptions](#)

C#

public class Interleaved2of5Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

Interleaved2of5Options

See Also

[Interleaved2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)


[Interleaved 2 of 5](#)

Interleaved2of5Options Members




Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Interleaved2of5Options](#) type.







Public Constructors

	Name	Description
	Interleaved2of5Options	Initializes a new instance of the Interleaved2of5Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarcodeGeneratorOptions)
	WideNarrowRatio	Gets or sets the wide bar to narrow bar ratio.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Interleaved2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)
[Interleaved 2 of 5](#)

Interleaved2of5Options Constructor

Initializes a new instance of the [Interleaved2of5Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Interleaved2of5Options()
```

See Also

[Interleaved2of5Options Class](#)

[Interleaved2of5Options Members](#)


[DevExpress.BarCodes Namespace](#)

Interleaved2of5Options Properties




Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Interleaved2of5Options](#) type.







Public Constructors

	Name	Description
	Interleaved2of5Options	Initializes a new instance of the Interleaved2of5Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarcodeGeneratorOptions)
	WideNarrowRatio	Gets or sets the wide bar to narrow bar ratio.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[Interleaved2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)
[Interleaved 2 of 5](#)

CalcChecksum Property

Gets or sets whether a check character should be calculated and added to the bar code data.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CalcChecksum As Boolean
```

C#

```
public bool CalcChecksum { get; set; }
```

Property Value

true, to add a check character; otherwise, **false**.

See Also

[Interleaved2of5Options Class](#)

[Interleaved2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)

WideNarrowRatio Property

Gets or sets the wide bar to narrow bar ratio.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property WideNarrowRatio As Single
```

C#

```
public Single WideNarrowRatio { get; set; }
```

Property Value

A [System.Single](#) value that specifies the density of bar code bars. The default is **3**.

Remarks

Use this property to specify the ratio of the thickest bar to the narrowest. The value should be equal to or greater than **2.5**.

See Also

[Interleaved2of5Options Class](#)

[Interleaved2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)

Matrix2of5Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class Matrix2of5Options Inherits [Industrial2of5Options](#)

C#

public class Matrix2of5Options : [Industrial2of5Options](#)

Inheritance Hierarchy

[Object](#)

[BarcodeGeneratorOptions](#)

[Industrial2of5Options](#)

Matrix2of5Options

See Also

[Matrix2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)

[Matrix 2 of 5](#)

Matrix2of5Options Members




Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [Matrix2of5Options](#) type.






Public Constructors

	Name	Description
	Matrix2of5Options	Initialize a new instance of the Matrix2of5Options class with default settings.

Public Properties

	Name	Description
	CalcChecksum	Gets or sets whether a check character should be calculated and added to the bar code data. (Inherited from Industrial2of5Options)
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	WideNarrowRatio	Gets or sets the wide bar to narrow bar ratio. (Inherited from Industrial2of5Options)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[Matrix2of5Options Members](#)[DevExpress.BarCodes Namespace](#)[Matrix 2 of 5](#)

Matrix2of5Options Constructor

Initialize a new instance of the [Matrix2of5Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Matrix2of5Options()
```

See Also

[Matrix2of5Options Class](#)

[Matrix2of5Options Members](#)

[DevExpress.BarCodes Namespace](#)

PDF417CompactionMode Enumeration

Lists compaction modes used in PDF417 symbology.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum PDF417CompactionMode
```

C#

```
public enum PDF417CompactionMode
```

Members

Name	Description
Binary	Allows encoding of all 8 -bit byte values.
Text	Allows encoding of all printable ASCII characters. The default mode.

See Also

[DevExpress.BarCodes Namespace](#)

PDF417ErrorLevel Enumeration

Lists levels of error correction.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum PDF417ErrorLevel
```

C#

```
public enum PDF417ErrorLevel
```

Members

Name	Description
Level0	Identifies the zero error correction level of the PDF417 bar code.
Level1	Identifies the first error correction level of the PDF417 bar code.
Level2	Identifies the second error correction level of the PDF417 bar code.
Level3	Identifies the third error correction level of the PDF417 bar code.
Level4	Identifies the fourth error correction level of the PDF417 bar code.
Level5	Identifies the fifth error correction level of the PDF417 bar code.
Level6	Identifies the sixth error correction level of the PDF417 bar code.
Level7	Identifies the seventh error correction level of the PDF417 bar code.
Level8	Identifies the eighth error correction level of the PDF417 bar code.

Remarks

Error correction allows the bar code to be partially damaged without resulting in a loss of data. A higher level of error correction means more redundant data in a bar code, so less useful data can be included, but the bar code will be more robust.

See Also

[DevExpress.BarCodes Namespace](#)

PDF417Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class PDF417Options Inherits [BarCodeGeneratorOptions](#)

C#

public class PDF417Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

PDF417Options

See Also

[PDF417Options Members](#)

[DevExpress.BarCodes Namespace](#)

[PDF417](#)

PDF417Options Members







Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [PDF417Options](#) type.



Public Constructors



	Name	Description
	PDF417Options	Initialize a new instance of the PDF417Options class with default settings.

Public Properties

	Name	Description
	Columns	Gets or sets the number of columns in the symbol in the data region.
	CompactionMode	Gets or sets the mode used for data encoding in the PDF417 Code.
	ErrorLevel	Gets or sets the error correction level for the PDF417 Code.
	Rows	Gets or sets the number of rows used to construct a PDF417 Code.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	TruncateSymbol	Gets or sets whether the right-hand side of the PDF417 Code symbol is truncated to create a compact code.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)

	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[PDF417Options Members](#)
[DevExpress.BarCodes Namespace](#)
[PDF417](#)

PDF417Options Constructor

Initialize a new instance of the [PDF417Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public PDF417Options()
```

See Also

[PDF417Options Class](#)

[PDF417Options Members](#)

[DevExpress.BarCodes Namespace](#)

PDF417Options Properties







Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [PDF417Options](#) type.



Public Constructors



	Name	Description
	PDF417Options	Initialize a new instance of the PDF417Options class with default settings.

Public Properties

	Name	Description
	Columns	Gets or sets the number of columns in the symbol in the data region.
	CompactionMode	Gets or sets the mode used for data encoding in the PDF417 Code.
	ErrorLevel	Gets or sets the error correction level for the PDF417 Code.
	Rows	Gets or sets the number of rows used to construct a PDF417 Code.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	TruncateSymbol	Gets or sets whether the right-hand side of the PDF417 Code symbol is truncated to create a compact code.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)

	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[PDF417Options Members](#)
[DevExpress.BarCodes Namespace](#)
[PDF417](#)

Columns Property

Gets or sets the number of columns in the symbol in the data region.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Columns As Integer
```

C#

```
public int Columns { get; set; }
```

Property Value

An integer that specifies the number of columns used in a PDF417 code. The default is **1**.

Remarks

The width of a PDF417 code image is determined by the [Barcode.Module](#) value that is the module width and the **Columns** value that is the number of columns.

See Also

[PDF417Options Class](#)

[PDF417Options Members](#)

[DevExpress.BarCodes Namespace](#)

CompactionMode Property

Gets or sets the mode used for data encoding in the PDF417 Code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CompactionMode As PDF417CompactionMode
```

C#

```
public PDF417CompactionMode CompactionMode { get; set; }
```

Property Value

A [PDF417CompactionMode](#) enumeration member that specifies the encoding algorithm.

See Also

[PDF417Options Class](#)

[PDF417Options Members](#)

[DevExpress.BarCodes Namespace](#)

ErrorLevel Property

Gets or sets the error correction level for the PDF417 Code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ErrorLevel As PDF417ErrorLevel
```

C#

```
public PDF417ErrorLevel ErrorLevel { get; set; }
```

Property Value

A [PDF417ErrorLevel](#) enumeration member that specifies the level of error correction.

Remarks

PDF417 uses Reed Solomon error correction to recover data if the code image has been damaged. The code with the higher error correction level is more robust. The default error correction level is **2**.

See Also

[PDF417Options Class](#)

[PDF417Options Members](#)

[DevExpress.BarCodes Namespace](#)

Rows Property

Gets or sets the number of rows used to construct a PDF417 Code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Rows As Integer
```

C#

```
public int Rows { get; set; }
```

Property Value

An integer that is the number of rows. The default is **3**.

Remarks

The height of a PDF417 Code is determined by the **Rows** value that is the number of rows in the code symbol.

See Also

[PDF417Options Class](#)

[PDF417Options Members](#)

[DevExpress.BarCodes Namespace](#)

TruncateSymbol Property

Gets or sets whether the right-hand side of the PDF417 Code symbol is truncated to create a compact code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property TruncateSymbol As Boolean
```

C#

```
public bool TruncateSymbol { get; set; }
```

Property Value

true, to create a compact bar code; otherwise, **false**. The default is **false**.

Remarks

Use the **TruncateSymbol** property to create a compact bar code by truncating its right-hand side. The following images demonstrate how the property value affects the bar code appearance.

TruncateSymbol = true	TruncateSymbol = false
	

See Also

[PDF417Options Class](#)

[PDF417Options Members](#)

[DevExpress.BarCodes Namespace](#)

PostNetOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class PostNetOptions Inherits [BarCodeGeneratorOptions](#)

C#

public class PostNetOptions : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

PostNetOptions

See Also

[PostNetOptions Members](#)

[DevExpress.BarCodes Namespace](#)

[PostNet](#)

PostNetOptions Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [PostNetOptions](#) type.







Public Constructors

	Name	Description
	PostNetOptions	Initialize a new instance of the PostNetOptions class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[PostNetOptions Members](#)
[DevExpress.BarCodes Namespace](#)
[PostNet](#)

PostNetOptions Constructor

Initialize a new instance of the [PostNetOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public PostNetOptions()
```

See Also

[PostNetOptions Class](#)

[PostNetOptions Members](#)

[DevExpress.BarCodes Namespace](#)

QRCodeCompactionMode Enumeration

Lists compaction modes for the QRCode bar code symbology.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum QRCodeCompactionMode
```

C#

```
public enum QRCodeCompactionMode
```

Members

Name	Description
AlphaNumeric	Encodes data from a set of alphanumeric characters, i.e., digits 0 - 9 ; upper case letters A - Z; other characters: space, (\$), (%), (*), (+), (-), (.), (/) and (:). The default mode.
Byte	Data is encoded at 8 bits per character.
Numeric	Encodes data from the set of digits from 0 to 9 .

See Also

[DevExpress.BarCodes Namespace](#)

QRCodeErrorLevel Enumeration

Lists error correction levels for the QRCode.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum QRCodeErrorLevel
```

C#

```
public enum QRCodeErrorLevel
```

Members

Name	Description
H	Allows recovery of up to 30% data loss.
L	Allows recovery of up to 7% data loss.
M	Allows recovery of up to 15% data loss.
Q	Allows recovery of up to 25% data loss.

Remarks

The data can be recovered if the QR Code image is partially damaged. To accomplish this, the QR Code uses Reed-Solomon error correction.

A higher correction level means a more robust bar code.

See Also

[DevExpress.BarCodes Namespace](#)

QRCodeOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class QRCodeOptions Inherits [BarCodeGeneratorOptions](#)

C#

public class QRCodeOptions : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

QRCodeOptions

See Also

[QRCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)


[QR Code](#)

QRCodeOptions Members





Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [QRCodeOptions](#) type.







Public Constructors

	Name	Description
	QRCodeOptions	Initialize a new instance of the QRCodeOptions class with default settings.

Public Properties

	Name	Description
	CompactionMode	Gets or sets the mode for data encoding in the QR code.
	ErrorLevel	Gets or sets the error correction level used in QR Code.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	Version	Gets or sets the QR Code version.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[QRCodeOptions Members](#)
[DevExpress.BarCodes Namespace](#)
[QR Code](#)

QRCodeOptions Constructor

Initialize a new instance of the [QRCodeOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public QRCodeOptions()
```

See Also

[QRCodeOptions Class](#)

[QRCodeOptions Members](#)


[DevExpress.BarCodes Namespace](#)

QRCodeOptions Properties





Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [QRCodeOptions](#) type.







Public Constructors

	Name	Description
	QRCodeOptions	Initialize a new instance of the QRCodeOptions class with default settings.

Public Properties

	Name	Description
	CompactionMode	Gets or sets the mode for data encoding in the QR code.
	ErrorLevel	Gets or sets the error correction level used in QR Code.
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)
	Version	Gets or sets the QR Code version.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[QRCodeOptions Members](#)
[DevExpress.BarCodes Namespace](#)
[QR Code](#)

CompactionMode Property

Gets or sets the mode for data encoding in the QR code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CompactionMode As QRCodeCompactionMode
```

C#

```
public QRCodeCompactionMode CompactionMode { get; set; }
```

Property Value

A [QRCodeCompactionMode](#) enumeration member that specifies the data compaction mode.

See Also

[QRCodeOptions Class](#)

[QRCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

ErrorLevel Property

Gets or sets the error correction level used in QR Code.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ErrorLevel As QRCodeErrorLevel
```

C#

```
public QRCodeErrorLevel ErrorLevel { get; set; }
```

Property Value

A [QRCodeErrorLevel](#) enumeration member that specifies the error correction level.

See Also

[QRCodeOptions Class](#)

[QRCodeOptions Members](#)

[DevExpress.BarCodes Namespace](#)

Version Property

Gets or sets the [QR Code](#) version.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property Version As [QRCodeVersion](#)

C#

public [QRCodeVersion](#) Version { get; set; }

Property Value

A [QRCodeVersion](#) enumeration value indicating the code version.

Remarks

The QR code version defines the size of a code's side.

The code snippet below shows how to use the **Version** property to specify the QR Code size:

C#	
<pre>Barcode barCode = new Barcode(); barCode.Symbology = Symbology.QRCode; barCode.Options.QRCode.Version = QRCodeVersion.Version10;</pre>	
Visual Basic	
<pre>Dim barCode As New Barcode() barCode.Symbology = Symbology.QRCode barCode.Options.QRCode.Version = QRCodeVersion.Version10</pre>	

See Also

- [QRCodeOptions Class](#)
- [QRCodeOptions Members](#)
- [DevExpress.BarCodes Namespace](#)

QRCodeVersion Enumeration

Lists values used to specify the [QR Code](#) version.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum QRCodeVersion
```

C#

```
public enum QRCodeVersion
```

Members

Name	Description
AutoVersion	The auto-calculated version.
Version1	Version 1 (21 x 21 modules)
Version10	Version 10 (57 x 57 modules)
Version11	Version 11 (61 x 61 modules).
Version12	Version 12 (65 x 65 modules).
Version13	Version 13 (69 x 69 modules).
Version14	Version 14 (73 x 73 modules).
Version15	Version 15 (77 x 77 modules).
Version16	Version 16 (81 x 81 modules).
Version17	Version 17 (85 x 85 modules).
Version18	Version 18 (89 x 89 modules).
Version19	Version 19 (93 x 93 modules).
Version2	Version 2 (25 x 25 modules).
Version20	Version 20 (97 x 97 modules).
Version21	Version 21 (101 x 101 modules).
Version22	Version 22 (105 x 105 modules).
Version23	Version 23 (109 x 109 modules).
Version24	Version 24 (113 x 113 modules).
Version25	Version 25 (117 x 117 modules).
Version26	Version 26 (121 x 121 modules).
Version27	Version 27 (125 x 125 modules).
Version28	Version 28 (129 x 129 modules).
Version29	Version 29 (133 x 133 modules).
Version3	Version 3 (29 x 29 modules).
Version30	Version 30 (137 x 137 modules).

Version31	Version 31 (141 x 141 modules).
Version32	Version 32 (145 x 145 modules).
Version33	Version 33 (149 x 149 modules).
Version34	Version 34 (153 x 153 modules).
Version35	Version 35 (157 x 157 modules).
Version36	Version 36 (161 x 161 modules).
Version37	Version 37 (165 x 165 modules).
Version38	Version 38 (169 x 169 modules).
Version39	Version 39 (173 x 173 modules).
Version4	Version 4 (33 x 33 modules).
Version40	Version 40 (177 x 177 modules).
Version5	Version 5 (37 x 37 modules).
Version6	Version 6 (41 x 41 modules).
Version7	Version 7 (45 x 45 modules).
Version8	Version 8 (49 x 49 modules)
Version9	Version 9 (53 x 53 modules).

Remarks

The values listed by the **QRCodeVersion** enumeration are used to set the [QRCodeOptions.Version](#) property. The QR Code version's data capacity depends on the amount of data, character type and error correction level.

See Also

[DevExpress.BarCodes Namespace](#)

Symbology Enumeration

Lists available symbologies (bar code types)

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Enum Symbology

C#

public enum Symbology


Members





Name	Description
Codabar	<div>A Codabar barcode with the alphanumeric string 0123456789+/-:.\$ printed below it.</div> <div>Codabar</div>
Code11	<div>A Code 11 barcode with the numeric string 0123456 printed below it.</div> <div>Code 11 (USD-8)</div>
Code128	<div>A Code 128 barcode with the text BarCode 0123456 printed below it.</div> <div>Code 128</div>

Code39	<div>A standard 1D barcode with vertical black bars of varying widths on a white background. Below the bars, the word "BARCODE" is printed in a bold, black, sans-serif font.</div> <div>Code 39 (USD-3)</div>
Code39Extended	<div>A standard 1D barcode with vertical black bars of varying widths on a white background. Below the bars, the word "BarCODE" is printed in a bold, black, sans-serif font.</div> <div>Code 39 Extended</div>
Code93	<div>A standard 1D barcode with vertical black bars of varying widths on a white background. Below the bars, the word "BARCODE" is printed in a bold, black, sans-serif font.</div> <div>Code 93</div>
Code93Extended	<div>A standard 1D barcode with vertical black bars of varying widths on a white background. Below the bars, the word "BarCode" is printed in a bold, black, sans-serif font.</div> <div>Code 93 Extended</div>

CodeMSI	<div></div> <div>MSI - Plessey</div>
DataBar	<div></div> <div>GS1 DataBar</div>
DataMatrix	<div></div> <div>Data Matrix (ECC200)</div>
DataMatrixGS1	<div></div> <div>GS1- Data Matrix</div>

EAN128	<div></div> <div>EAN-128 (UCC)</div>
EAN13	<div></div> <div>EAN 13</div>
EAN8	<div></div> <div>EAN 8</div>
Industrial2of5	<div></div> <div>Industrial 2 of 5</div>

IntelligentMail	<div></div> <div>Intelligent Mail</div>
Interleaved2of5	<div></div> <div>Interleaved 2 of 5</div>
ITF14	<div></div> <div>GS1 DataBar</div>
Matrix2of5	<div></div> <div>Matrix 2 of 5</div>
PDF417	<div></div> <div>PDF417</div>

PostNet	<div><p>300001</p></div> <div>PostNet</div>
QRCode	<div></div>
UPCA	<div><p>785342354638</p></div> <div>UPC-A</div>
UPCE0	<div><p>04252614</p></div> <div>UPC-E0</div>

UPCE1	<div><p>A barcode with the number 1 425261 1 printed below it.</p></div> <div>UPC-E1</div>
UPCSupplemental2	<div><p>A barcode with the number 9 785318 001116 01 printed below it. The text 'EAN 13' is in red and 'UPC Supplemental 2' is in blue.</p></div> <div>UPC Supplemental 2</div>
UPCSupplemental5	<div><p>A barcode with the number 9 785318 001116 01234 printed below it. The text 'EAN 13' is in red and 'UPC Supplemental 5' is in blue.</p></div> <div>UPC Supplemental 5</div>

See Also
[DevExpress.BarCodes Namespace](#)

UPCAOptions Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class UPCAOptions Inherits [EAN13Options](#)

C#

public class UPCAOptions : [EAN13Options](#)

Inheritance Hierarchy

[Object](#)

[BarcodeGeneratorOptions](#)

[EAN13Options](#)

UPCAOptions

 Derived classes

See Also

[UPCAOptions Members](#)

[DevExpress.BarCodes Namespace](#)


[UPC-A](#)

UPCAOptions Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [UPCAOptions](#) type.







Public Constructors

	Name	Description
	UPCAOptions	Initialize a new instance of the UPCAOptions class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[UPCAOptions Members](#)
[DevExpress.BarCodes Namespace](#)
[UPC-A](#)

UPCAOptions Constructor

Initialize a new instance of the [UPCAOptions](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public UPCAOptions()
```

See Also

[UPCAOptions Class](#)

[UPCAOptions Members](#)

[DevExpress.BarCodes Namespace](#)

UPCE0Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class UPCE0Options Inherits UPCAOptions
```

C#

```
public class UPCE0Options : UPCAOptions
```

Inheritance Hierarchy

[Object](#)

[BarcodeGeneratorOptions](#)

[EAN13Options](#)

[UPCAOptions](#)

UPCE0Options

See Also

[UPCE0Options Members](#)

[DevExpress.BarCodes Namespace](#)

[UPC-E0](#)

UPCE0Options Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [UPCE0Options](#) type.







Public Constructors

	Name	Description
	UPCE0Options	Initialize a new instance of the UPCE0Options class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[UPCE0Options Members](#)
[DevExpress.BarCodes Namespace](#)
[UPC-E0](#)

UPCE0Options Constructor

Initialize a new instance of the [UPCE0Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public UPCE0Options()
```

See Also

[UPCE0Options Class](#)

[UPCE0Options Members](#)

[DevExpress.BarCodes Namespace](#)

UPCE1Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class UPCE1Options Inherits [UPCAOptions](#)

C#

```
public class UPCE1Options : UPCAOptions
```

Inheritance Hierarchy

[Object](#)

[BarcodeGeneratorOptions](#)

[EAN13Options](#)

[UPCAOptions](#)

UPCE1Options

See Also

[UPCE1Options Members](#)

[DevExpress.BarCodes Namespace](#)

[UPC-E1](#)

UPCE1Options Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [UPCE1Options](#) type.







Public Constructors

	Name	Description
	UPCE1Options	Initialize a new instance of the UPCE1Options class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[UPCE1Options Members](#)
[DevExpress.BarCodes Namespace](#)
[UPC-E1](#)

UPCE1Options Constructor

Initialize a new instance of the [UPCE1Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public UPCE1Options()
```

See Also

[UPCE1Options Class](#)

[UPCE1Options Members](#)

[DevExpress.BarCodes Namespace](#)

UPCSupplemental2Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class UPCSupplemental2Options Inherits [BarCodeGeneratorOptions](#)

C#

public class UPCSupplemental2Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

UPCSupplemental2Options

See Also

[UPCSupplemental2Options Members](#)

[DevExpress.BarCodes Namespace](#)


[UPC Supplemental 2](#)

UPCSupplemental2Options Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [UPCSupplemental2Options](#) type.







Public Constructors

	Name	Description
	UPCSupplemental2Options	Initialize a new instance of the UPCSupplemental2Options class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[UPCSupplemental2Options Members](#)
[DevExpress.BarCodes Namespace](#)
[UPC Supplemental 2](#)

UPCSupplemental2Options Constructor

Initialize a new instance of the [UPCSupplemental2Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public UPCSupplemental2Options()
```

See Also

[UPCSupplemental2Options Class](#)

[UPCSupplemental2Options Members](#)

[DevExpress.BarCodes Namespace](#)

UPCSupplemental5Options Class

Contains options specific to this symbology (bar code type).

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class UPCSupplemental5Options Inherits [BarCodeGeneratorOptions](#)

C#

public class UPCSupplemental5Options : [BarCodeGeneratorOptions](#)

Inheritance Hierarchy

[Object](#)

[BarCodeGeneratorOptions](#)

UPCSupplemental5Options

See Also

[UPCSupplemental5Options Members](#)

[DevExpress.BarCodes Namespace](#)


[UPC Supplemental 5](#)

UPCSupplemental5Options Members


Contains options specific to this symbology (bar code type).

The following tables list the members exposed by the [UPCSupplemental5Options](#) type.







Public Constructors

	Name	Description
	UPCSupplemental5Options	Initialize a new instance of the UPCSupplemental5Options class with default settings.

Public Properties

	Name	Description
	ShowCodeText	Gets or sets whether bar code text data should be displayed in a bar code image. (Inherited from BarCodeGeneratorOptions)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[UPCSupplemental5Options Members](#)
[DevExpress.BarCodes Namespace](#)
[UPC Supplemental 5](#)

UPCSupplemental5Options Constructor

Initialize a new instance of the [UPCSupplemental5Options](#) class with default settings.

Namespace: [DevExpress.BarCodes](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public UPCSupplemental5Options()
```

See Also

[UPCSupplemental5Options Class](#)

[UPCSupplemental5Options Members](#)

[DevExpress.BarCodes Namespace](#)

DevExpress.Compression

Contains classes of the Compression library intended to implement zip compression and archive generation functionality.

Classes

	Class	Description
	AllowFileOverwriteEventArgs	Provides data for the <code>ZipArchive.AllowFileOverwrite</code> event.
	CanContinueEventArgs	Base class that provides data to indicate whether the process can proceed further.
	ErrorEventArgs	Provides data for the Error event.
	ProgressEventArgs	Provides data for the <code>ZipArchive.Progress</code> event.
	WrongPasswordException	An exception that is thrown when an encrypted zip file item is extracted with an invalid password.
	ZipArchive	The central object of the library - a package of entries containing compressed data.
	ZipArchiveOptionsBehavior	Contains options that specify how the zip archive performs certain actions.
	ZipByteArrayItem	A zip item specific to the byte array source.
	ZipDirectoryItem	A zip item specific to the directory.
	ZipFileItem	A zip item specific to the file source.
	ZipItem	An entry in the zip archive containing compressed data.
	ZipItemAddingEventArgs	Provides data for the <code>ZipArchive.ItemAdding</code> event.
	ZipStreamItem	A zip item specific to the stream source.
	ZipTextItem	A zip item specific to the text source.

Delegates

	Delegate	Description
	AllowFileOverwriteEventHandler	A method that will handle the AllowFileOverwrite event.
	ErrorEventHandler	A method that will handle the Error event.
	ProgressEventHandler	A method that will handle the Progress event.
	ZipItemAddingEventHandler	A method that will handle the ItemAdding event.

Enumerations

	Enumeration	Description
--	-------------	-------------

	AllowFileOverwriteMode	Lists available modes to handle a file conflict when unzipping the archive.
	EncryptionType	Lists the available encryption types.
	ZipItemAddingAction	Lists a possible action when the ZipArchive.ItemAdding event is handled.

AllowFileOverwriteEventArgs Class

Provides data for the ZipArchive.AllowFileOverwrite event.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class AllowFileOverwriteEventArgs Inherits [CancelEventArgs](#)

C#

```
public class AllowFileOverwriteEventArgs : CancelEventArgs
```

Remarks

Handle the [ZipArchive.AllowFileOverwrite](#) event to manually resolve the file conflict when unzipping.

Inheritance Hierarchy

[Object](#)

[EventArgs](#)

[CancelEventArgs](#)

AllowFileOverwriteEventArgs

See Also

[AllowFileOverwriteEventArgs Members](#)


[DevExpress.Compression Namespace](#)

AllowFileOverwriteEventArgs Members




Provides data for the ZipArchive.AllowFileOverwrite event.

The following tables list the members exposed by the [AllowFileOverwriteEventArgs](#) type.







Public Constructors

	Name	Description
	AllowFileOverwriteEventArgs	Initializes a new instance of the AllowFileOverwriteEventArgs class with the zip item and the path to where it will be unzipped.


Public Properties

	Name	Description
	Cancel	Gets or sets a value indicating whether the event should be canceled. (Inherited from CancelEventArgs)
	TargetFilePath	Gets the path to the file to which the zip item will be unzipped.
	ZipItem	Obtains the zip item for which a file conflict is detected.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
	Empty	Represents an event with no event data. (Inherited from EventArgs)

See Also
[AllowFileOverwriteEventArgs Members](#)
[DevExpress.Compression Namespace](#)

AllowFileOverwriteEventArgs Constructor

Initializes a new instance of the [AllowFileOverwriteEventArgs](#) class with the zip item and the path to where it will be unzipped.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal zipItem As ZipItem,  
    ByVal targetFilePath As String  
)
```

C#

```
public AllowFileOverwriteEventArgs(  
    ZipItem zipItem,  
    string targetFilePath  
)
```

Parameters

zipItem

A [ZipItem](#) descendant that is the zip item being unzipped.

targetFilePath

A string that specifies the path to the file in which the zip item will be saved after unzipping.

Remarks

The instance of the [AllowFileOverwriteEventArgs](#) class is automatically created, initialized and passed to the [ZipArchive.AllowFileOverwrite](#) event handler.

See Also

[AllowFileOverwriteEventArgs Class](#)

[AllowFileOverwriteEventArgs Members](#)


[DevExpress.Compression Namespace](#)

AllowFileOverwriteEventArgs Properties




Provides data for the ZipArchive.AllowFileOverwrite event.

The following tables list the members exposed by the [AllowFileOverwriteEventArgs](#) type.







Public Constructors

	Name	Description
	AllowFileOverwriteEventArgs	Initializes a new instance of the AllowFileOverwriteEventArgs class with the zip item and the path to where it will be unzipped.

Public Properties

	Name	Description
	Cancel	Gets or sets a value indicating whether the event should be canceled. (Inherited from CancelEventArgs)
	TargetFilePath	Gets the path to the file to which the zip item will be unzipped.
	ZipItem	Obtains the zip item for which a file conflict is detected.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
	Empty	Represents an event with no event data. (Inherited from EventArgs)

See Also

[AllowFileOverwriteEventArgs Members](#)
[DevExpress.Compression Namespace](#)

TargetFilePath Property

Gets the path to the file to which the zip item will be unzipped.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property TargetFilePath As String
```

C#

```
public string TargetFilePath { get; }
```

Property Value

A string that is the file path.

See Also

[AllowFileOverwriteEventArgs Class](#)

[AllowFileOverwriteEventArgs Members](#)

[DevExpress.Compression Namespace](#)

ZipItem Property

Obtains the zip item for which a file conflict is detected.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property ZipItem As ZipItem
```

C#

```
public ZipItem ZipItem { get; }
```

Property Value

A [ZipItem](#) descendant that is the zip item being unzipped.

See Also

[AllowFileOverwriteEventArgs Class](#)

[AllowFileOverwriteEventArgs Members](#)

[DevExpress.Compression Namespace](#)

AllowFileOverwriteEventHandler Delegate

A method that will handle the [ZipArchive.AllowFileOverwrite](#) event.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Delegate Sub AllowFileOverwriteEventHandler(  
    ByVal sender As Object,  
    ByVal e As AllowFileOverwriteEventArgs  
)
```

C#

```
public delegate void AllowFileOverwriteEventHandler(  
    object sender,  
    AllowFileOverwriteEventArgs e  
)
```

Parameters

The declaration of your event handler must have the same parameters as the **AllowFileOverwriteEventHandler** delegate declaration.

sender

The event source. This parameter identifies the [ZipArchive](#), which raised the event.

e

A [AllowFileOverwriteEventArgs](#) object which contains event data.

Remarks

When creating a [AllowFileOverwriteEventHandler](#) delegate, you identify the method that will handle the corresponding event. To associate an event with your event handler, add a delegate instance to this event. The event handler is called whenever the event occurs unless you remove the delegate. For more information about event handler delegates, see **Events and Delegates** in **MSDN**.

See Also

[DevExpress.Compression Namespace](#)

AllowFileOverwriteMode Enumeration

Lists available modes to handle a file conflict when unzipping the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum AllowFileOverwriteMode
```

C#

```
public enum AllowFileOverwriteMode
```

Members

Name	Description
Allow	Allows overwriting files in the destination path without prompting.
Custom	Fires the ZipArchive.AllowFileOverwrite event if a file name conflict occurs.
Forbidden	If a file name conflict occurs, the unzipping process skips to the next zip item and the file in the destination path remains intact.

Remarks

If the [ZipArchive.Extract](#) or the [ZipItem.Extract](#) method tries to create a file with a name that already exists, a file conflict occurs. You can specify the [ZipArchiveOptionsBehavior.AllowFileOverwrite](#) property to handle the conflict automatically.

See Also

[DevExpress.Compression Namespace](#)

CanContinueEventArgs Class

Base class that provides data to indicate whether the process can proceed further.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class CanContinueEventArgs Inherits [EventArgs](#)

C#

```
public abstract class CanContinueEventArgs : EventArgs
```

Inheritance Hierarchy

[Object](#)

[EventArgs](#)

CanContinueEventArgs

Derived classes

See Also

[CanContinueEventArgs Members](#)


[DevExpress.Compression Namespace](#)

CanContinueEventArgs Members







Base class that provides data to indicate whether the process can proceed further.

The following tables list the members exposed by the [CanContinueEventArgs](#) type.


Public Properties

	Name	Description
	CanContinue	Gets or sets the value that specifies whether the process can proceed further.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
	Empty	Represents an event with no event data. (Inherited from EventArgs)

See Also


[CanContinueEventArgs Members](#)

[DevExpress.Compression Namespace](#)







CanContinueEventArgs Properties

Base class that provides data to indicate whether the process can proceed further.
The following tables list the members exposed by the [CanContinueEventArgs](#) type.


Public Properties

	Name	Description
	CanContinue	Gets or sets the value that specifies whether the process can proceed further.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
	Empty	Represents an event with no event data. (Inherited from EventArgs)

See Also
[CanContinueEventArgs Members](#)
[DevExpress.Compression Namespace](#)

CanContinue Property

Gets or sets the value that specifies whether the process can proceed further.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property CanContinue As [Boolean](#)

C#

```
public bool CanContinue { get; set; }
```

Property Value

true, to allow the process to continue; otherwise, **false**.

Remarks

The [ZipArchive.Progress](#) and the [ZipArchive.Error](#) event handlers allow you to stop the archive creation by setting the **CanContinue** property of the respective event arguments to **false**.

See Also

[CanContinueEventArgs Class](#)

[CanContinueEventArgs Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.Progress](#)

[ZipArchive.Error](#)

EncryptionType Enumeration

Lists the available encryption types.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum EncryptionType`
C#
`public enum EncryptionType`

Members

Name	Description
Aes128	Use the Advanced Encryption Standard (AES) cryptographic with keys that are 128 bits long.
Aes192	Use the Advanced Encryption Standard (AES) cryptographic with keys that are 192 bits long.
Aes256	Use the Advanced Encryption Standard (AES) cryptographic with keys that are 256 bits long.
None	Do not use encryption.
PkZip	Use the PKZIP encryption algorithm.

Remarks

Use the [ZipItem.EncryptionType](#) to set the required encryption for the [ZipItem](#).

See Also
[DevExpress.Compression Namespace](#)

ErrorEventArgs Class

Provides data for the [ZipArchive.Error](#) event.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class ErrorEventArgs Inherits [CanContinueEventArgs](#)

C#

public class ErrorEventArgs : [CanContinueEventArgs](#)

Inheritance Hierarchy

[Object](#)

[EventArgs](#)

[CanContinueEventArgs](#)

ErrorEventArgs

See Also

[ErrorEventArgs Members](#)

[DevExpress.Compression Namespace](#)

ErrorEventArgs Members



Provides data for the [ZipArchive.Error](#) event.

The following tables list the members exposed by the [ErrorEventArgs](#) type.








Public Constructors

	Name	Description
	ErrorEventArgs	Initializes a new instance of the ErrorEventArgs class with the specified settings.


Public Properties

	Name	Description
	CanContinue	Gets or sets the value that specifies whether the process can proceed further. (Inherited from CanContinueEventArgs)
	ItemName	Obtains the zip item name for which the error occurs.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetException	Obtains the exception that triggered the error event.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
	Empty	Represents an event with no event data. (Inherited from EventArgs)

See Also
[ErrorEventArgs Members](#)
[DevExpress.Compression Namespace](#)

ErrorEventArgs Constructor

Initializes a new instance of the [ErrorEventArgs](#) class with the specified settings.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal exception As Exception,  
    ByVal itemName As String  
)
```

C#

```
public ErrorEventArgs(  
    Exception exception,  
    string itemName  
)
```

Parameters

exception

A [System.Exception](#) object that is the exception that triggered the event.

itemName

A string that is the name of the zip item that caused an error or an empty string if an error occurs while reading the directory or saving the archive.

See Also

[ErrorEventArgs Class](#)

[ErrorEventArgs Members](#)


[DevExpress.Compression Namespace](#)

ErrorEventArgs Properties


Provides data for the [ZipArchive.Error](#) event.

The following tables list the members exposed by the [ErrorEventArgs](#) type.








Public Constructors

	Name	Description
	ErrorEventArgs	Initializes a new instance of the ErrorEventArgs class with the specified settings.


Public Properties

	Name	Description
	CanContinue	Gets or sets the value that specifies whether the process can proceed further. (Inherited from CanContinueEventArgs)
	ItemName	Obtains the zip item name for which the error occurs.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetException	Obtains the exception that triggered the error event.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
	Empty	Represents an event with no event data. (Inherited from EventArgs)

See Also
[ErrorEventArgs Members](#)
[DevExpress.Compression Namespace](#)

ItemName Property

Obtains the zip item name for which the error occurs.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property ItemName As String
```

C#

```
public string ItemName { get; }
```

Property Value

A string that is the zip item name.

Remarks

The zip item name is the [ZipItem.Name](#) value. The **ItemName** allows you to identify the problematic zip item and decide whether it can be safely skipped.

If the [ZipArchive](#) object cannot process a directory containing files or it is unable to save the resulting archive, the **ItemName** property value is empty. In this situation, you should analyze the exception that triggered an error by calling the [GetException](#) method.

See Also

[ErrorEventArgs Class](#)

[ErrorEventArgs Members](#)

[DevExpress.Compression Namespace](#)

ErrorEventArgs Methods

Provides data for the [ZipArchive.Error](#) event.

The following tables list the members exposed by the [ErrorEventArgs](#) type.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetException	Obtains the exception that triggered the error event.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[ErrorEventArgs Members](#)

[DevExpress.Compression Namespace](#)

GetException Method

Obtains the exception that triggered the error event.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetException() As Exception
```

C#

```
public Exception GetException()
```

Return Value

A [System.Exception](#) object that is the exception that triggered the event.

See Also

[ErrorEventArgs Class](#)

[ErrorEventArgs Members](#)

[DevExpress.Compression Namespace](#)

ErrorEventHandler Delegate

A method that will handle the **Error** event of the ZipArchive.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Delegate Sub ErrorHandler(  
    ByVal sender As Object,  
    ByVal args As EventArgs  
)
```

C#

```
public delegate void ErrorHandler(  
    object sender,  
    EventArgs args  
)
```

Parameters

The declaration of your event handler must have the same parameters as the **ErrorEventHandler** delegate declaration.

sender
The event source. This parameter identifies the [ZipArchive](#) which raised the event.

args
A [EventArgs](#) object which contains event data.

Remarks

When creating a [ErrorHandler](#) delegate, you identify the method that will handle the corresponding event. To associate an event with your event handler, add a delegate instance to this event. The event handler is called whenever the event occurs unless you remove the delegate. For more information about event handler delegates, see **Events and Delegates** in **MSDN**.

See Also

[DevExpress.Compression Namespace](#)

ProgressEventArgs Class

Provides data for the ZipArchive.Progress event.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class ProgressEventArgs Inherits [CanContinueEventArgs](#)

C#

```
public class ProgressEventArgs : CanContinueEventArgs
```

Remarks

Handle the [Progress](#) event to control the archiving process or to interrupt it when required.

Inheritance Hierarchy

[Object](#)

[EventArgs](#)

[CanContinueEventArgs](#)

ProgressEventArgs

See Also

[ProgressEventArgs Members](#)

[DevExpress.Compression Namespace](#)

ProgressEventArgs Members



Provides data for the ZipArchive.Progress event.

The following tables list the members exposed by the [ProgressEventArgs](#) type.







Public Constructors

	Name	Description
	ProgressEventArgs	Initializes a new instance of the ProgressEventArgs class with the specified progress value.

Public Properties

	Name	Description
	CanContinue	Gets or sets the value that specifies whether the process can proceed further. (Inherited from CanContinueEventArgs)
	Progress	Obtains the progress value.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
--	------	-------------

	Empty	Represents an event with no event data. (Inherited from EventArgs)
--	-------	---

See Also
[ProgressEventArgs Members](#)
[DevExpress.Compression Namespace](#)

ProgressEventArgs Constructor

Initializes a new instance of the [ProgressEventArgs](#) class with the specified progress value.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal progress As Double  
)
```

C#

```
public ProgressEventArgs(  
    Double progress  
)
```

Parameters

progress

A [System.Double](#) value that specifies the progress.

Remarks

The instance of the [ProgressEventArgs](#) class is automatically created, initialized and passed to the [ZipArchive.Progress](#) event handler.

See Also

[ProgressEventArgs Class](#)

[ProgressEventArgs Members](#)

[DevExpress.Compression Namespace](#)

ProgressEventArgs Properties

Provides data for the ZipArchive.Progress event.
The following tables list the members exposed by the [ProgressEventArgs](#) type.







Public Constructors

	Name	Description
	ProgressEventArgs	Initializes a new instance of the ProgressEventArgs class with the specified progress value.

Public Properties

	Name	Description
	CanContinue	Gets or sets the value that specifies whether the process can proceed further. (Inherited from CanContinueEventArgs)
	Progress	Obtains the progress value.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
--	------	-------------

	Empty	Represents an event with no event data. (Inherited from EventArgs)
--	-------	---

See Also
[ProgressEventArgs Members](#)
[DevExpress.Compression Namespace](#)

Progress Property

Obtains the progress value.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Progress As Double
```

C#

```
public Double Progress { get; }
```

Property Value

A [System.Double](#) value in the range from 0 to 1 that indicates the progress.

See Also

[ProgressEventArgs Class](#)

[ProgressEventArgs Members](#)

[DevExpress.Compression Namespace](#)

ProgressEventHandler Delegate

A method that will handle the [ZipArchive.Error](#) event.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Delegate Sub ProgressEventHandler(  
    ByVal sender As Object,  
    ByVal args As ProgressEventArgs  
)
```

C#

```
public delegate void ProgressEventHandler(  
    object sender,  
    ProgressEventArgs args  
)
```

Parameters

The declaration of your event handler must have the same parameters as the **ProgressEventHandler** delegate declaration.

sender
The event source. This parameter identifies the [ZipArchive](#), which raised the event.

args

A [ErrorEventArgs](#) object which contains event data.

Remarks

When creating a [ProgressEventHandler](#) delegate, you identify the method that will handle the corresponding event. To associate an event with your event handler, add a delegate instance to this event. The event handler is called whenever the event occurs unless you remove the delegate. For more information about event handler delegates, see **Events and Delegates** in **MSDN**.

See Also

[DevExpress.Compression Namespace](#)

WrongPasswordException Class

An exception that is thrown when an encrypted zip file item is extracted with an invalid password.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class WrongPasswordException Inherits [Exception](#)

C#

```
public class WrongPasswordException : Exception
```

Remarks

If a [ZipItem.Password](#) property for the encrypted zip item contains an invalid password, the call to the [ZipItem.Extract](#) method fires the **WrongPasswordException** exception. Wrap the [ZipItem.Extract](#) method call in try..catch statements to intercept the exception.

Inheritance Hierarchy

[Object](#)

[Exception](#)

WrongPasswordException

See Also

[WrongPasswordException Members](#)


[DevExpress.Compression Namespace](#)

WrongPasswordException Members








An exception that is thrown when an encrypted zip file item is extracted with an invalid password.

The following tables list the members exposed by the [WrongPasswordException](#) type.



Public Constructors

	Name	Description
	WrongPasswordException	Initializes a new instance of the WrongPasswordException class with the specified password.

Public Properties

	Name	Description
	Data	Gets a collection of key/value pairs that provide additional, user-defined information about the exception. (Inherited from Exception)
	HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
	HResult	(Inherited from Exception)
	InnerException	Gets the System.Exception instance that caused the current exception. (Inherited from Exception)
	Message	Gets a message that describes the current exception. (Inherited from Exception)
	Password	Gets the name of the incorrect password which caused an exception.
	Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
	StackTrace	Gets a string representation of the frames on the call stack at the time the current exception was thrown. (Inherited from Exception)
	TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current

		System.Object . (Inherited from Object)
	GetBaseException	When overridden in a derived class, returns the System.Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetObjectData	When overridden in a derived class, sets the System.Runtime.Serialization.SerializationInfo with information about the exception. (Inherited from Exception)
	GetType	Gets the runtime type of the current instance. (Inherited from Exception)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Creates and returns a string representation of the current exception. (Inherited from Exception)

See Also[WrongPasswordException Members](#)[DevExpress.Compression Namespace](#)

WrongPasswordException Constructor

Initializes a new instance of the [WrongPasswordException](#) class with the specified password.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal password As String  
)
```

C#

```
public WrongPasswordException(  
    string password  
)
```

Parameters

password

A string that is the incorrect password which is the reason for the exception.

See Also

[WrongPasswordException Class](#)

[WrongPasswordException Members](#)


[DevExpress.Compression Namespace](#)

WrongPasswordException Properties










An exception that is thrown when an encrypted zip file item is extracted with an invalid password.

The following tables list the members exposed by the [WrongPasswordException](#) type.



Public Constructors

	Name	Description
	WrongPasswordException	Initializes a new instance of the WrongPasswordException class with the specified password.

Public Properties

	Name	Description
	Data	Gets a collection of key/value pairs that provide additional, user-defined information about the exception. (Inherited from Exception)
	HelpLink	Gets or sets a link to the help file associated with this exception. (Inherited from Exception)
	HResult	(Inherited from Exception)
	InnerException	Gets the System.Exception instance that caused the current exception. (Inherited from Exception)
	Message	Gets a message that describes the current exception. (Inherited from Exception)
	Password	Gets the name of the incorrect password which caused an exception.
	Source	Gets or sets the name of the application or the object that causes the error. (Inherited from Exception)
	StackTrace	Gets a string representation of the frames on the call stack at the time the current exception was thrown. (Inherited from Exception)
	TargetSite	Gets the method that throws the current exception. (Inherited from Exception)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current

		System.Object . (Inherited from Object)
	GetBaseException	When overridden in a derived class, returns the System.Exception that is the root cause of one or more subsequent exceptions. (Inherited from Exception)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetObjectData	When overridden in a derived class, sets the System.Runtime.Serialization.SerializationInfo with information about the exception. (Inherited from Exception)
	GetType	Gets the runtime type of the current instance. (Inherited from Exception)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Creates and returns a string representation of the current exception. (Inherited from Exception)

See Also[WrongPasswordException Members](#)[DevExpress.Compression Namespace](#)

Password Property

Gets the name of the incorrect password which caused an exception.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Password As String
```

C#

```
public string Password { get; }
```

Property Value

A string that is the incorrect password.

Remarks

Use the **try.. catch** statement when calling the [ZipArchive.Extract](#) or the [ZipItem.Extract](#) method to catch the [WrongPasswordException](#) exception and inform the end-user that the password is incorrect.

See Also

[WrongPasswordException Class](#)

[WrongPasswordException Members](#)

[DevExpress.Compression Namespace](#)

ZipArchive Class

The central object of the library - a package of entries containing compressed data.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class ZipArchive Inherits Object, IZipItemOwner,  
    IEnumerable(of ZipItem),  
    IEnumerable,  
    IDisposable
```

C#

```
public class ZipArchive : object, IZipItemOwner,  
    IEnumerable<ZipItem>,  
    IEnumerable,  
    IDisposable
```

Remarks

The **ZipArchive** is a package of entries which are the [ZipItem](#) descendants. To access an individual entry by its index or a name, use the [Item](#) property.

If you create a new archive, create the **ZipArchive** instance by using a constructor. To ensure that the zip archive instance will be correctly disposed of, call the constructor within the **using** statement. When a new instance of the zip archive is created, you can add entries by calling the methods specific for each source type.

Source Type	Method	Entry Type
File	AddFile	ZipFileItem
Directory	AddDirectory	ZipFileItem
Stream	AddStream	ZipStreamItem
Text	AddText	ZipTextItem
Array of Bytes	AddByteArray	ZipByteArrayItem

To extract archive content, use the static method [Read](#). The method creates a **ZipArchive** instance, and you can call the [Extract](#) method for the entire archive or the [ZipItem.Extract](#) method for each zip item.

To update the archive, create an instance of the **ZipArchive** class using the [Read](#) method, modify it as required and save it using the [Save](#) method.

You can control the process of adding files to the archive by handling the [ItemAdding](#) event. To control the process of compressing and saving the archive (and to interrupt it when required), handle the [Progress](#) event.

Inheritance Hierarchy

[Object](#)
ZipArchive

See Also


[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[Examples](#)

ZipArchive Members






The central object of the library - a package of entries containing compressed data.

The following tables list the members exposed by the [ZipArchive](#) type.








Public Constructors

	Name	Description
	ZipArchive	Overloaded. Initializes a new instance of the ZipArchive class with default settings.

Public Properties





	Name	Description
	CatchExceptions	Allows you to set an application-wide flag that specifies whether exceptions which occur in ZipArchive operations are caught.
	Count	Obtains the number of zip items in the archive.
	EncryptionType	Gets or sets the default encryption type.
	FileName	Reserved for future use.
	Item	Overloaded. Provides indexed access to an individual zip item in the archive.
	OptionsBehavior	Provides access to archive options.
	Password	Gets or sets the password for an encrypted archive.

Public Methods

	Name	Description
	AddByteArray	Creates a zip item from a byte array and adds it to archive.
	AddDirectory	Overloaded. Recursively add files and directories to the archive.
	AddFile	Overloaded. Creates a zip file item for the specified file and adds it to the archive.
	AddFiles	Overloaded. Adds files to archive.
	AddStream	Creates a zip stream item and adds it to the archive.
	AddText	Overloaded. Creates an empty text zip item and adds it to archive.
	Dispose	Clears all zip items, disposes all associated streams and releases all

		resources used by the ZipArchive object.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Extract	Overloaded. Extract all archive items as files into the current directory.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	IsValid	Reserved for future use.
	Read	Overloaded. Static method that creates a ZipArchive instance from the archive file, uses the specified encoding for the zip item names and allows you not to catch exceptions when extracting a particular zip entry.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveItem	Overloaded. Deletes a specified zip item from the archive.
	Save	Overloaded. Compresses data and saves it to a specified stream.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
	UpdateDirectory	Overloaded. Recursively updates files and nested directories starting from the specified directory.
	UpdateFile	Overloaded. Updates content of the zip file item.
	UpdateStream	Updates the content of the zip stream item.
	UpdateText	Overloaded. Updates the content of the zip text item and specifies a new character encoding.
	Validate	Verifies archive entries by extracting them into memory and catching exceptions.

Public Events

	Name	Description
	AllowFileOverwrite	Occurs when the item extracted from the archive tries to overwrite a file that already exists.
	Error	Fires when an error occurs while adding files to the archive, processing archive items or saving the archive.
	ItemAdding	Occurs before a zip item is added to the archive.
	Progress	Occurs evenly while the items are being compressed to indicate progress.

See Also[ZipArchive Members](#)[DevExpress.Compression Namespace](#)[Examples](#)

ZipArchive Constructor

Initializes a new instance of the [ZipArchive](#) class with default settings.

Overload List

Name	Description
ZipArchive()	Initializes a new instance of the ZipArchive class with default settings.
ZipArchive(bool catchExceptions)	Initializes a new instance of the ZipArchive class and specifies whether to catch exceptions in archive operations.

- See Also**
- [ZipArchive Class](#)
 - [ZipArchive Members](#)
 - [DevExpress.Compression Namespace](#)
 - [ZipArchive Overload List](#)

ZipArchive Constructor

Initializes a new instance of the [ZipArchive](#) class with default settings.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public ZipArchive()
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive Overload List](#)

ZipArchive Constructor (bool)

Initializes a new instance of the [ZipArchive](#) class and specifies whether to catch exceptions in archive operations.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal catchExceptions As Boolean  
)
```

C#

```
public ZipArchive(  
    bool catchExceptions  
)
```

Parameters

catchExceptions

True, to catch exceptions during archive operations; otherwise, **false**.

Remarks

You can also use the [CatchExceptions](#) static property and [Read](#) method overrides to change the archive behavior regarding exceptions. When exceptions are caught by default, you can handle the [Error](#) event to analyze them and perform the required actions.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)


[ZipArchive Overload List](#)

ZipArchive Properties







The central object of the library - a package of entries containing compressed data.

The following tables list the members exposed by the [ZipArchive](#) type.








Public Constructors

	Name	Description
	ZipArchive	Overloaded. Initializes a new instance of the ZipArchive class with default settings.

Public Properties





	Name	Description
	CatchExceptions	Allows you to set an application-wide flag that specifies whether exceptions which occur in ZipArchive operations are caught.
	Count	Obtains the number of zip items in the archive.
	EncryptionType	Gets or sets the default encryption type.
	FileName	Reserved for future use.
	Item	Overloaded. Provides indexed access to an individual zip item in the archive.
	OptionsBehavior	Provides access to archive options.
	Password	Gets or sets the password for an encrypted archive.

Public Methods

	Name	Description
	AddByteArray	Creates a zip item from a byte array and adds it to archive.
	AddDirectory	Overloaded. Recursively add files and directories to the archive.
	AddFile	Overloaded. Creates a zip file item for the specified file and adds it to the archive.
	AddFiles	Overloaded. Adds files to archive.
	AddStream	Creates a zip stream item and adds it to the archive.
	AddText	Overloaded. Creates an empty text zip item and adds it to archive.
	Dispose	Clears all zip items, disposes all associated streams and releases all

		resources used by the ZipArchive object.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Extract	Overloaded. Extract all archive items as files into the current directory.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	IsValid	Reserved for future use.
	Read	Overloaded. Static method that creates a ZipArchive instance from the archive file, uses the specified encoding for the zip item names and allows you not to catch exceptions when extracting a particular zip entry.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveItem	Overloaded. Deletes a specified zip item from the archive.
	Save	Overloaded. Compresses data and saves it to a specified stream.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
	UpdateDirectory	Overloaded. Recursively updates files and nested directories starting from the specified directory.
	UpdateFile	Overloaded. Updates content of the zip file item.
	UpdateStream	Updates the content of the zip stream item.
	UpdateText	Overloaded. Updates the content of the zip text item and specifies a new character encoding.
	Validate	Verifies archive entries by extracting them into memory and catching exceptions.

Public Events

	Name	Description
	AllowFileOverwrite	Occurs when the item extracted from the archive tries to overwrite a file that already exists.
	Error	Fires when an error occurs while adding files to the archive, processing archive items or saving the archive.
	ItemAdding	Occurs before a zip item is added to the archive.
	Progress	Occurs evenly while the items are being compressed to indicate progress.

See Also[ZipArchive Members](#)[DevExpress.Compression Namespace](#)[Examples](#)

CatchExceptions Property

Allows you to set an application-wide flag that specifies whether exceptions which occur in ZipArchive operations are caught.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Property CatchExceptions As Nullable(of Boolean)
```

C#

```
public static Nullable<Boolean> CatchExceptions { get; set; }
```

Property Value

True to catch exceptions; **false** to allow exceptions to propagate; if **null** or not set, no effect.

Remarks

You can explicitly set the **CatchExceptions** property to override the behavior specified in [Read](#) method calls and in the [ZipArchive](#) constructor. Once set, it affects all ZipArchive instances. If **true**, exceptions are intercepted, if **false**, exceptions are allowed to propagate.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

Count Property

Obtains the number of zip items in the archive.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Count As Integer
```

C#

```
public int Count { get; }
```

Property Value

An integer that is the number of zip items.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

EncryptionType Property

Gets or sets the default encryption type.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property EncryptionType As EncryptionType
```

C#

```
public EncryptionType EncryptionType { get; set; }
```

Property Value

An [EncryptionType](#) enumeration member specifying the encryption type. Default is [EncryptionType.None](#).

Remarks

When a password is set for the archive, the encryption type is automatically set to [EncryptionType.PkZip](#).

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

FileName Property

Reserved for future use.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property FileName As String
```

C#

```
public string FileName { get; set; }
```

Property Value

A string that is the file name. By default is **null**.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

Item Property

Provides access to individual zip items in the archive by their names.

Overload List

Name	Description
ZipItem this[string name] { get; }	Provides access to individual zip items in the archive by their names.
ZipItem this[int index] { get; }	Provides indexed access to an individual zip item in the archive.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.Item Overload List](#)

Provides access to individual zip items in the archive by their names.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
Public ReadOnly Property Item(
 ByVal name As [String](#)
) As [ZipItem](#)

C#
public [ZipItem](#) this[
 [string](#) name
] { get; }

Parameters
name
A string that is the name of the zip item.

Property Value
A [ZipItem](#) descendant that is the zip item with the specified name.

Remarks
The **Item** searches for the zip item by its [ZipItem.Name](#) value.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.Item Overload List](#)

Provides indexed access to an individual zip item in the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
Public ReadOnly Property Item(
 ByVal index As [Integer](#)
) As [ZipItem](#)
C#

```
public ZipItem this[  
    int index  
] { get; }
```

Parameters*index*

An integer value that is the zero-based index of items within the archive.

Property Value

A [ZipItem](#) descendant that is the archive item at the specified position within the collection of items in the archive.

Remarks

Use this property to access individual zip items using index notation.

See Also[ZipArchive Class](#)[ZipArchive Members](#)[DevExpress.Compression Namespace](#)[ZipArchive.Item Overload List](#)

OptionsBehavior Property

Provides access to archive options.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property OptionsBehavior As ZipArchiveOptionsBehavior
```

C#

```
public ZipArchiveOptionsBehavior OptionsBehavior { get; }
```

Property Value

A [ZipArchiveOptionsBehavior](#) object containing archive options.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

Password Property

Gets or sets the password for an encrypted archive.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Password As String
```

C#

```
public string Password { get; set; }
```

Property Value

A string that is the password used to encrypt the archive.

Remarks

If the **Password** is empty, the encryption is not performed when the archive is saved using the [Save](#) method. When the **Password** is set to a non-empty value and the [ZipItem.EncryptionType](#) is **null**, the [ZipItem.EncryptionType](#) is automatically set to the [EncryptionType.PkZip](#) value.

To extract the content of the encrypted zip item, specify its **Password** value. If the password is incorrect, the [WrongPasswordException](#) fires when the [ZipItem.Extract](#) method (or the [Extract](#) method) is called.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[WrongPasswordException](#)

ZipArchive Events

The central object of the library - a package of entries containing compressed data.

The following tables list the members exposed by the [ZipArchive](#) type.

Public Events

	Name	Description
	AllowFileOverwrite	Occurs when the item extracted from the archive tries to overwrite a file that already exists.
	Error	Fires when an error occurs while adding files to the archive, processing archive items or saving the archive.
	ItemAdding	Occurs before a zip item is added to the archive.
	Progress	Occurs evenly while the items are being compressed to indicate progress.

See Also
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[Examples](#)

AllowFileOverwrite Event

Occurs when the item extracted from the archive tries to overwrite a file that already exists.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public AllowFileOverwrite As AllowFileOverwriteEventHandler
```

C#

```
public event AllowFileOverwriteEventHandler AllowFileOverwrite
```

Event Data

The event handler receives an argument of type [AllowFileOverwriteEventArgs](#) containing data related to this event.

The following **AllowFileOverwriteEventArgs** properties provide information specific to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
TargetFilePath	Gets the path to the file to which the zip item will be unzipped.
ZipItem	Obtains the zip item for which a file conflict is detected.

Remarks

By handling the **AllowFileOverwrite** event, you can decide whether a certain file should be replaced with an item extracted from the archive. This functionality can be useful for implementing backup synchronization schemes.

To leave the conflicting file intact and skip to the next zip item, set the **Cancel** property of the event arguments to **true**.

The following code illustrates how to implement the requirement that the conflicting target file should be overwritten only if the corresponding file in the archive has been accessed no later than an hour ago.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example illustrates how to handle a file name conflict when files are extracted from archive. If a file with the same name exists, the [AllowFileOverwrite](#) event occurs. You can handle this event and determine that if the file in the folder is newer than the zip item, the file in the folder should not be overwritten.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void UnzipArchiveConflict() {
    string pathToZipArchive = "Documents\\Example.zip";
    string pathToExtract = "Documents\\!Extracted";
    using (ZipArchive archive = ZipArchive.Read(pathToZipArchive)) {
        archive.OptionsBehavior.AllowFileOverwrite = AllowFileOverwriteMode.Custom;
        archive.AllowFileOverwrite += archive_AllowFileOverwrite;
        foreach (ZipItem item in archive) {
            item.Extract(pathToExtract);
        }
    }

    private void archive_AllowFileOverwrite(object sender, AllowFileOverwriteEventArgs e) {
        FileInfo fi = new FileInfo(e.TargetFilePath);
        if (e.ZipItem.LastWriteTime < fi.LastWriteTime) e.Cancel = true;
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression
Public Sub UnzipArchiveConflict()
    Dim pathToZipArchive As String = "Documents\Example.zip"
    Dim pathToExtract As String = "Documents\!Extracted"
    Using archive As ZipArchive = ZipArchive.Read(pathToZipArchive)
        archive.OptionsBehavior.AllowFileOverwrite = AllowFileOverwriteMode.Custom
        AddHandler archive.AllowFileOverwrite, AddressOf archive_AllowFileOverwrite
        For Each item As ZipItem In archive
            item.Extract(pathToExtract)
        Next item
    End Using
End Sub
Private Sub archive_AllowFileOverwrite(ByVal sender As Object, ByVal e As AllowFileOverwriteEventArgs)
    Dim fi As New FileInfo(e.TargetFilePath)
    If e.ZipItem.LastWriteTime < fi.LastWriteTime Then
        e.Cancel = True
    End If
End Sub
End Sub
```

See Also[ZipArchive Class](#)[ZipArchive Members](#)[DevExpress.Compression Namespace](#)

Error Event

Fires when an error occurs while adding files to the archive, processing archive items or saving the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

event Public Error As [ErrorEventHandler](#)

C#

public event [ErrorEventHandler](#) Error

Event Data

The event handler receives an argument of type [ErrorEventArgs](#) containing data related to this event.

The following **ErrorEventArgs** properties provide information specific to this event.

Property	Description
CanContinue	Gets or sets the value that specifies whether the process can proceed further.
ItemName	Obtains the zip item name for which the error occurs.

Remarks

Use the [ErrorEventArgs.GetException](#) method of the event arguments to determine the exception that is the reason for the error. The [ErrorEventArgs.ItemName](#) property value is the name of the [ZipItem](#) for which an error occurs. If an error occurs when adding a directory to an archive or saving the compressed archive, the name of the ZipItem is empty.

By handling the **Error** event, you can decide whether to cancel processing a failed item and skip to the next item; otherwise, you can cancel processing the entire archive by setting the [CanContinueEventArgs.CanContinue](#) property to **false**.

Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=E4695 .	

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveDirectoryWithError() {
    string path = this.startupPath;
    using (ZipArchive archive = new ZipArchive()) {
        archive.Error += archive_Error;
        archive.AddDirectory(path);
        archive.Save("Documents\\ArchiveDirectoryWithError.zip");
    }
}

private void archive_Error(object sender, DevExpress.Compression.ErrorEventArgs args) {
    string errorMessage;
    Exception e = args.GetException();
    if (String.IsNullOrEmpty(args.ItemName)) {
        errorMessage = e.Message;
    }
    else {
        errorMessage = String.Format("Item: {0}\\n\\nDescription:\\n{1}", args.ItemName, e.Message);
    }
    string descriptionMessage = "Click Cancel to abort operation. Click OK to skip the item and co
    string message = String.Format("{0}\\n{1}", errorMessage, descriptionMessage);
    if (MessageBox.Show(message, "Error", MessageBoxButtons.OKCancel) == DialogResult.Cancel) {
        args.CanContinue = false;
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveDirectoryWithError()
    Dim path As String = Me.startupPath
    Using archive As New ZipArchive()
        AddHandler archive.Error, AddressOf archive_Error
        archive.AddDirectory(path)
        archive.Save("Documents\ArchiveDirectoryWithError.zip")
    End Using
End Sub

Private Sub archive_Error(ByVal sender As Object, ByVal args As DevExpress.Compression.ErrorEventArgs)
    Dim errorMessage As String
    Dim e As Exception = args.GetException()
    If String.IsNullOrEmpty(args.ItemName) Then
        errorMessage = e.Message
    Else
        errorMessage = String.Format("Item: {0}" & Constants.vbLf + Constants.vbLf & "Description: {0}", e.Message, args.ItemName)
    End If
    Dim descriptionMessage As String = "Click Cancel to abort operation. Click OK to skip the item."
    Dim message As String = String.Format("{0}" & Constants.vbLf & "{1}", errorMessage, descriptionMessage)
    If MessageBox.Show(message, "Error", MessageBoxButtons.OKCancel) = DialogResult.Cancel Then
        args.CanContinue = False
    End If
End Sub

End Sub
```

See Also[ZipArchive Class](#)[ZipArchive Members](#)[DevExpress.Compression Namespace](#)

ItemAdding Event

Occurs before a zip item is added to the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

event Public ItemAdding As [ZipItemAddingEventHandler](#)

C#

public event [ZipItemAddingEventHandler](#) ItemAdding

Event Data

The event handler receives an argument of type [ZipItemAddingEventArgs](#) containing data related to this event. The following **ZipItemAddingEventArgs** properties provide information specific to this event.

Property	Description
Action	Gets or sets the action required to perform after the ItemAdding event is handled.
Item	Obtains the zip item being added to the archive.

Remarks

By handling the **ItemAdding** event you can decide whether or not a certain item should be added to the archive. The item is accessible by using the [ZipItemAddingEventArgs.Item](#) property. The [ZipItemAddingEventArgs.Action](#) property allows you to skip the current item or to stop adding items to the archive.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example demonstrates how to handle the [ItemAdding](#) event to decide for each file whether it should be included in the archive.

If a file creation date is not the current date, the file is not added to the archive. A volatile variable is used to indicate whether the process should be stopped - it can be useful to interrupt archive creation if too many files are specified.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;
    volatile bool stopArchiving = false;
    public void FilterArchiveFiles() {
        string[] sourcefiles = this.sourceFiles;
        using (ZipArchive archive = new ZipArchive()) {
            archive.ItemAdding += archive_ItemAdding;
            foreach (string file in sourceFiles) {
                archive.AddFile(file, "/");
            }
            archive.Save("Documents\\FilterArchiveFiles.zip");
        }
    }
    private void archive_ItemAdding(object sender, ZipItemAddingEventArgs args) {
        if (args.Item.CreationTime.Date != DateTime.Today)
            args.Action = ZipItemAddingAction.Cancel;
        if (stopArchiving) args.Action = ZipItemAddingAction.Stop;
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression
'INSTANT VB TODO TASK: There is no VB.NET equivalent to 'volatile':
'ORIGINAL LINE: volatile bool stopArchiving = False;
Private stopArchiving As Boolean = False
Public Sub FilterArchiveFiles()
    Dim sourcefiles() As String = Me.sourceFiles
    Using archive As New ZipArchive()
        AddHandler archive.ItemAdding, AddressOf archive_ItemAdding
        For Each file As String In Me.sourceFiles
            archive.AddFile(file, "/" )
        Next file
        archive.Save("Documents\FilterArchiveFiles.zip")
    End Using
End Sub
Private Sub archive_ItemAdding(ByVal sender As Object, ByVal args As ZipItemAddingEventArgs)
    If args.Item.CreationTime.Date <> DateTime.Today Then
        args.Action = ZipItemAddingAction.Cancel
    End If
    If stopArchiving Then
        args.Action = ZipItemAddingAction.Stop
    End If
End Sub
End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

Progress Event

Occurs evenly while the items are being compressed to indicate progress.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

event Public Progress As [ProgressEventHandler](#)

C#

public event [ProgressEventHandler](#) Progress

Event Data

The event handler receives an argument of type [ProgressEventArgs](#) containing data related to this event.

The following **ProgressEventArgs** properties provide information specific to this event.

Property	Description
CanContinue	Gets or sets the value that specifies whether the process can proceed further.
Progress	Obtains the progress value.

Remarks

By handling the **Progress** event, you can obtain information about the progress of zipping the archive items. You can also interrupt archive creation by setting the [CanContinueEventArgs.CanContinue](#) value to **false**. After that, the archive is saved with all items which have been zipped to that point and the process is finished.

The following code snippet illustrates how you can use the volatile variable to interrupt archive creation at any time. When the **Progress** event occurs, the event handler checks for the **stopProgress** value. If it is **true**, the process stops.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

volatile bool stopProgress = false;
public void CancelArchiveProgress() {
    string[] sourcefiles = this.sourceFiles;
    using (ZipArchive archive = new ZipArchive()) {
        archive.Progress += archive_Progress;
        foreach (string file in sourceFiles) {
            archive.AddFile(file, "/");
        }
        archive.Save("Documents\\CancelArchiveProgress.zip");
    }
}

private void archive_Progress(object sender, ProgressEventArgs args) {
    args.CanContinue = !this.stopProgress;
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression
'INSTANT VB TODO TASK: There is no VB.NET equivalent to 'volatile':
'ORIGINAL LINE: volatile bool stopProgress = False;
    Private stopProgress As Boolean = False
    Public Sub CancelArchiveProgress()
        Dim sourcefiles() As String = Me.sourceFiles
        Using archive As New ZipArchive()
            AddHandler archive.Progress, AddressOf archive_Progress
            For Each file As String In Me.sourceFiles
                archive.AddFile(file, "/")
            Next file
            archive.Save("Documents\CancelArchiveProgress.zip")
        End Using
    End Sub
    Private Sub archive_Progress(ByVal sender As Object, ByVal args As ProgressEventArgs)
        args.CanContinue = Not Me.stopProgress
    End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)









ZipArchive Methods

The central object of the library - a package of entries containing compressed data.

The following tables list the members exposed by the [ZipArchive](#) type.

Public Methods

	Name	Description
	AddByteArray	Creates a zip item from a byte array and adds it to archive.
	AddDirectory	Overloaded. Recursively add files and directories to the archive.
	AddFile	Overloaded. Creates a zip file item for the specified file and adds it to the archive.
	AddFiles	Overloaded. Adds files to archive.
	AddStream	Creates a zip stream item and adds it to the archive.
	AddText	Overloaded. Creates an empty text zip item and adds it to archive.
	Dispose	Clears all zip items, disposes all associated streams and releases all resources used by the ZipArchive object.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Extract	Overloaded. Extract all archive items as files into the current directory.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	IsValid	Reserved for future use.
	Read	Overloaded. Static method that creates a ZipArchive instance from the archive file, uses the specified encoding for the zip item names and allows you not to catch exceptions when extracting a particular zip entry.
	ReferenceEquals	Determines whether the specified System.Object instances are the same

		instance. (Inherited from Object)
	RemoveItem	Overloaded. Deletes a specified zip item from the archive.
	Save	Overloaded. Compresses data and saves it to a specified stream.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
	UpdateDirectory	Overloaded. Recursively updates files and nested directories starting from the specified directory.
	UpdateFile	Overloaded. Updates content of the zip file item.
	UpdateStream	Updates the content of the zip stream item.
	UpdateText	Overloaded. Updates the content of the zip text item and specifies a new character encoding.
	Validate	Verifies archive entries by extracting them into memory and catching exceptions.

See Also[ZipArchive Members](#)[DevExpress.Compression Namespace](#)[Examples](#)

AddByteArray Method

Creates a zip item from a byte array and adds it to archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddByteArray(  
    ByVal name As String,  
    ByVal content As Byte[]  
) As ZipByteArrayItem
```

C#

```
public ZipByteArrayItem AddByteArray(  
    string name,  
    Byte[] content  
)
```

Parameters

- name*
A string that is the name of the zip item.
- content*
A System.Byte[] array to compress and store in a zip item.

Return Value

A [ZipByteArrayItem](#) object that is the compressed array of bytes in an archive.

Remarks

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This code snippet adds a byte array to an archive as an item with the name "myByteArray" and outputs zipped data to the stream.

C#

```
(ZipExamples.cs)  
using DevExpress.Compression;  
public void ArchiveByteArray() {  
    byte[] myByteArray = Enumerable.Repeat((byte)0x78, 10000).ToArray();  
    using (Stream myZippedStream = new FileStream("Documents\\ArchiveByteArray.zip", FileMode.Create))  
        using (ZipArchive archive = new ZipArchive()) {  
            archive.AddByteArray("myByteArray", myByteArray);  
            archive.Save(myZippedStream);  
        }  
}
```

Visual Basic

```
(ZipExamples.vb)  
Imports DevExpress.Compression  
Public Sub ArchiveByteArray()  
    Dim myByteArray() As Byte = Enumerable.Repeat(CByte(&H78), 10000).ToArray()  
    Using myZippedStream As Stream = New FileStream("Documents\\ArchiveByteArray.zip", FileMode.Create)  
        Using archive As New ZipArchive()  
            archive.AddByteArray("myByteArray", myByteArray)  
            archive.Save(myZippedStream)  
        End Using  
    End Using  
End Sub
```

See Also[ZipArchive Class](#)[ZipArchive Members](#)[DevExpress.Compression Namespace](#)

AddDirectory Method

Recursively add files and directories to the archive directory named by the last component of the specified path.

Overload List

Name	Description
ZipDirectoryItem AddDirectory(string path)	Recursively add files and directories to the archive directory named by the last component of the specified path.
ZipDirectoryItem AddDirectory(string path, string archivePath)	Recursively add files and directories to the archive.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.AddDirectory Overload List](#)

Recursively add files and directories to the archive directory named by the last component of the specified path.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddDirectory(  
    ByVal path As String  
) As ZipDirectoryItem
```

C#

```
public ZipDirectoryItem AddDirectory(  
    string path  
)
```

Parameters

path
A string that is the path to a directory which will be recursively included in the archive.

Return Value

A [ZipDirectoryItem](#) that is the zip item container for the directory. Currently, the method always returns **null**.

Remarks

The **AddDirectory** recursively collects all files and folders starting from the folder specified by the **path** parameter. The tree of directories and files is added to the archive directory named by the last component of a specified path.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.AddDirectory Overload List](#)

Recursively add files and directories to the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddDirectory(  
    ByVal path As String,  
    ByVal archivePath As String  
) As ZipDirectoryItem
```

C#

```
public ZipDirectoryItem AddDirectory(  
    string path,  
    string archivePath  
)
```

Parameters

path

A string that is the path to a directory which will be included in the archive.

archivePath

A string that is the path in the archive.

Return Value

A [ZipDirectoryItem](#) that is the zip item container for the directory. Currently, the method always returns **null**.

Remarks

The **AddDirectory**

recursively collects all files and folders starting from the folder specified by the **path** parameter. The tree of directories and files is added to the archive directory specified by the **archivePath** parameter. If the **archivePath** parameter is set to "/", the tree is added to the archive root. If the **archivePath** parameter is **null**, the last component of a path is used as the name of the archive directory in which a tree is built .

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.AddDirectory Overload List](#)

AddFile Method

Creates a zip file item for the specified file and adds it to the archive.

Overload List

Name	Description
ZipFileItem.AddFile(string fileName)	Creates a zip file item for the specified file and adds it to the archive.
ZipFileItem.AddFile(string fileName, string relativePath)	Creates a zip file item for the specified file and adds it to the specified path in the archive.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.AddFile Overload List](#)

Creates a zip file item for the specified file and adds it to the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddFile(  
    ByVal fileName As String  
) As ZipFileItem
```

C#

```
public ZipFileItem AddFile(  
    string fileName  
)
```

Parameters

fileName
A string that is the path to the file.

Return Value

A [ZipFileItem](#) that is the file item in the archive.

Remarks

This method adds the specified file to the archive. The path in the archive is constructed from the file path so that if the file path is rooted, the archive path is the path to the root; otherwise, the path in archive is composed of the same components as the file path. It means that the file "C:\Temp\AAA\BBB\winnipooh.txt" is added to the archive as a [ZipFileItem](#) with the [ZipItem.Name](#) that equals "Temp/AAA/BBB/winnipooh.txt" (archive path) and a [ZipFileItem.FileName](#) that equals "C:\Temp\AAA\BBB\winnipooh.txt" (file path).

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This code snippet demonstrates how to create a new archive and add files to the archive root. The [Save](#) method compresses data and saves the archive to a file.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveFiles() {
    string[] sourcefiles = this.sourceFiles;
    using (ZipArchive archive = new ZipArchive()) {
        foreach (string file in sourcefiles) {
            archive.AddFile(file, "/");
        }
        archive.Save("Documents\\ArchiveFiles.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveFiles()
    Dim sourcefiles() As String = Me.sourceFiles
    Using archive As New ZipArchive()
        For Each file As String In sourcefiles
            archive.AddFile(file, "/")
        Next file
        archive.Save("Documents\\ArchiveFiles.zip")
    End Using
End Sub
```

- See Also**
- [ZipArchive Class](#)
 - [ZipArchive Members](#)
 - [DevExpress.Compression Namespace](#)
 - [ZipArchive.AddFile Overload List](#)

Creates a zip file item for the specified file and adds it to the specified path in the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddFile(  
    ByVal fileName As String,  
    ByVal relativePath As String  
) As ZipFileItem
```

C#

```
public ZipFileItem AddFile(  
    string fileName,  
    string relativePath  
)
```

Parameters

- fileName*
A string that is the path to the file.
- relativePath*
A string that is the path in the archive.

Return Value

A [ZipFileItem](#) that is the file item in the archive.

Remarks

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=E4695>.

This code snippet demonstrates how to create a new archive and add files to the archive root. The [Save](#) method compresses data and saves the archive to a file.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveFiles() {
    string[] sourcefiles = this.sourceFiles;
    using (ZipArchive archive = new ZipArchive()) {
        foreach (string file in sourcefiles) {
            archive.AddFile(file, "/");
        }
        archive.Save("Documents\\ArchiveFiles.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveFiles()
    Dim sourcefiles() As String = Me.sourceFiles
    Using archive As New ZipArchive()
        For Each file As String In sourcefiles
            archive.AddFile(file, "/")
        Next file
        archive.Save("Documents\\ArchiveFiles.zip")
    End Using
End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.AddFile Overload List](#)

AddFiles Method

Adds files to archive.

Overload List

Name	Description
void AddFiles(IEnumerable<String> fileNames)	Adds files to archive.
void AddFiles(IEnumerable<String> fileNames, string archivePath)	Adds files to the archive at the specified path.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.AddFiles Overload List](#)

Adds files to archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AddFiles(  
    ByVal fileNames As IEnumerable(of String)  
)
```

C#

```
public void AddFiles(  
    IEnumerable<String> fileNames  
)
```

Parameters

fileNames
A list of file names which implements the System.Collections.Generic.IEnumerable<System.String> interface.

Remarks

The **AddFiles** method enables you to batch add files to the archive. The archive path to which the files will be added is constructed from the file path by discarding the root path component, i.e., the file path "C:\Temp\test.txt" will be the "Temp/test.txt".

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example recursively enumerates all files that have a .txt extension, reads each line of the file, and if a line contains the string "DevExpress", the file is added to archive.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveFilesBatch() {
    string path = this.startupPath;
    using (ZipArchive archive = new ZipArchive()) {
        var files = from file in System.IO.Directory.EnumerateFiles(path, "*.xml",
            System.IO.SearchOption.AllDirectories)
            from line in System.IO.File.ReadLines(file)
            where line.Contains("DevExpress")
            select file;
        archive.AddFiles(files);
        archive.Save("Documents\\ArchiveFilesBatch.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveFilesBatch()
    Dim path As String = Me.startupPath
    Using archive As New ZipArchive()
        Dim files = _
            From file In System.IO.Directory.EnumerateFiles(path, "*.xml", System.IO.SearchOption
            Where line.Contains("DevExpress") _
            Select file
        archive.AddFiles(files)
        archive.Save("Documents\\ArchiveFilesBatch.zip")
    End Using
End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.AddFiles Overload List](#)

Adds files to the archive at the specified path.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AddFiles(
    ByVal fileNames As IEnumerable(of String),
    ByVal archivePath As String
)
```

C#

```
public void AddFiles(
    IEnumerable<String> fileNames,
    string archivePath
)
```

Parameters

fileNames

A list of file names which implements the System.Collections.Generic.IEnumerable<System.String> interface.

archivePath

A string that specifies the archive path to which the files will be added.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.AddFiles Overload List](#)

AddStream Method

Creates a zip stream item and adds it to the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddStream(  
    ByVal name As String,  
    ByVal stream As Stream  
) As ZipStreamItem
```

C#

```
public ZipStreamItem AddStream(  
    string name,  
    Stream stream  
)
```

Parameters

- name*
A string that is the name of the newly created zip item.
- stream*
A [System.IO.Stream](#) object containing data to add to the archive.

Return Value

A [ZipStreamItem](#) object that references the stream of data included in an archive.

Remarks

The **AddStream** method does not make a copy of original steam. For proper operation, you should ensure that the stream is not closed until the archive is saved using the [Save](#) method. After the archive is saved, you have to close and dispose of the stream yourself.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

C#

```
(ZipExamples.cs)  
using DevExpress.Compression;  
    public void ArchiveStream() {  
        using (Stream myStream = new MemoryStream(System.Text.Encoding.UTF8.GetBytes("DevExpress"))) {  
            using (Stream myZippedStream = new FileStream("Documents\\ArchiveStream.zip", System.IO.FileMode.Create)) {  
                using (ZipArchive archive = new ZipArchive()) {  
                    archive.AddStream("myStream", myStream);  
                    archive.Save(myZippedStream);  
                }  
            }  
        }  
    }  
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression
Public Sub ArchiveStream()
    Using myStream As Stream = New MemoryStream(System.Text.Encoding.UTF8.GetBytes("DevExpress"))
        Using myZippedStream As Stream = New FileStream("Documents\ArchiveStream.zip", System.IO.FileMode.Append)
            Using archive As New ZipArchive()
                archive.AddStream("myStream", myStream)
                archive.Save(myZippedStream)
            End Using
        End Using
    End Using
End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

AddText Method

Creates an empty text zip item and adds it to archive.

Overload List

Name	Description
ZipTextItem AddText(string name)	Creates an empty text zip item and adds it to archive.
ZipTextItem AddText(string name, string content)	Creates a text zip item and adds it to the archive.
ZipTextItem AddText(string name, string content, Encoding encoding)	Creates a text zip item and adds it to archive.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.AddText Overload List](#)

Creates an empty text zip item and adds it to archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddText(  
    ByVal name As String  
) As ZipTextItem
```

C#

```
public ZipTextItem AddText(  
    string name  
)
```

Parameters

name
A string specifying the name of the text zip item.

Return Value

A [ZipTextItem](#) object that is the compressed text item in an archive.

Remarks

Default character encoding is UTF-8.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example creates a [ZipTextItem](#) with the name "Text_DE.txt". The content of the zip item is the text string. When unzipped, the archive creates a Text_DE.txt file in the working directory containing the specified text. Since the [ZipTextItem.ContentEncoding](#) is not specified, the default, UTF-8, is used.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveText() {
    using (ZipArchive archive = new ZipArchive()) {
        archive.AddText("Text_DE.txt", "Komprimieren großer Dateien mühelos");
        archive.Save("Documents\\ArchiveText.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveText()
    Using archive As New ZipArchive()
        archive.AddText("Text_DE.txt", "Komprimieren großer Dateien mühelos")
        archive.Save("Documents\\ArchiveText.zip")
    End Using
End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.AddText Overload List](#)

Creates a text zip item and adds it to the archive.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddText(
    ByVal name As String,
    ByVal content As String
) As ZipTextItem
```

C#

```
public ZipTextItem AddText(
    string name,
    string content
)
```

Parameters

name

A string specifying the name of the text zip item.

content

A string that is the text to compress and store in a text zip item.

Return Value

A [ZipTextItem](#) object that is the compressed text item in an archive.

Remarks

Default character encoding is UTF8.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.AddText Overload List](#)

Creates a text zip item and adds it to archive.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddText(  
    ByVal name As String,  
    ByVal content As String,  
    ByVal encoding As Encoding  
) As ZipTextItem
```

C#

```
public ZipTextItem AddText(  
    string name,  
    string content,  
    Encoding encoding  
)
```

Parameters

name

A string that is the name of the text zip item.

content

A string that is the text to compress and store in a text zip item.

encoding

A [System.Text.Encoding](#) object that specifies character encoding of the compressed text.

Return Value

A [ZipTextItem](#) object that is the compressed text item in an archive.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.AddText Overload List](#)

Dispose Method

Clears all zip items, disposes all associated streams and releases all resources used by the **ZipArchive** object.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Dispose()
```

C#

```
public void Dispose()
```

Remarks

Call the **Dispose** method when you have called the [Save](#) or [Extract](#) method of the **ZipArchive** object instance and are finished working with it. This releases all resources allocated by the object.

You are advised to operate with the [ZipArchive](#) object instance within the **using** statement (**Using** block in Visual Basic).

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

Extract Method

Extract all archive items as files into the current directory.

Overload List

Name	Description
void Extract()	Extract all archive items as files into the current directory.
void Extract(string path)	Extracts all archive items as files into the specified directory.
void Extract(string path, AllowFileOverwriteMode mode)	Extracts all archive items as files into the specified directory and enables you to define the behavior in case of file name conflicts.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.Extract Overload List](#)

Extract all archive items as files into the current directory.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
Public Sub Extract()
C#
public void Extract()

Remarks

If the archive contains directories, the directory structure will be recreated in the current directory, and the current directory is considered to be the root directory. The [ZipItem.Name](#) property value becomes the file name of the extracted file.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example illustrates how to load the zip file and extract it to the specified directory. If the target directory is not specified, the current directory is the target.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void UnzipArchive() {
    string pathToZipArchive = "Documents\\Example.zip";
    string pathToExtract = "Documents\\!Extracted";
    using (ZipArchive archive = ZipArchive.Read(pathToZipArchive)) {
        foreach (ZipItem item in archive) {
            item.Extract(pathToExtract);
        }
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression
Public Sub UnzipArchive()
    Dim pathToZipArchive As String = "Documents\Example.zip"
    Dim pathToExtract As String = "Documents\!Extracted"
    Using archive As ZipArchive = ZipArchive.Read(pathToZipArchive)
        For Each item As ZipItem In archive
            item.Extract(pathToExtract)
        Next item
    End Using
End Sub
```

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.Extract Overload List](#)

Extracts all archive items as files into the specified directory.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Extract(  
    ByVal path As String  
)
```

C#


```
public void Extract(  
    string path  
)
```

Parameters

path
A string that is the path to the directory to which the files are extracted.

Remarks

If the archive contains directories, the directory structure will be recreated in the specified directory. The [ZipItem.Name](#) property value becomes the file name of the extracted file.

 Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=E4695 .	

This example illustrates how to load the zip file and extract it to the specified directory. If the target directory is not specified, the current directory is the target.

C#	
<pre>(ZipExamples.cs) using DevExpress.Compression; public void UnzipArchive() { string pathToZipArchive = "Documents\\Example.zip"; string pathToExtract = "Documents\\!Extracted"; using (ZipArchive archive = ZipArchive.Read(pathToZipArchive)) { foreach (ZipItem item in archive) { item.Extract(pathToExtract); } } }</pre>	

Visual Basic	
---------------------	--

```
(ZipExamples.vb)
Imports DevExpress.Compression
Public Sub UnzipArchive()
    Dim pathToZipArchive As String = "Documents\Example.zip"
    Dim pathToExtract As String = "Documents\!Extracted"
    Using archive As ZipArchive = ZipArchive.Read(pathToZipArchive)
        For Each item As ZipItem In archive
            item.Extract(pathToExtract)
        Next item
    End Using
End Sub
```

- See Also**
- [ZipArchive Class](#)
 - [ZipArchive Members](#)
 - [DevExpress.Compression Namespace](#)
 - [ZipArchive.Extract Overload List](#)

Extracts all archive items as files into the specified directory and enables you to define the behavior in case of file name conflicts.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Extract(  
    ByVal path As String,  
    ByVal mode As AllowFileOverwriteMode  
)
```

C#


```
public void Extract(  
    string path,  
    AllowFileOverwriteMode mode  
)
```

Parameters

- path*
A string that is the path to the directory to which the files are extracted.
- mode*
An [AllowFileOverwriteMode](#) enumeration member that specifies the behavior if a file name conflict occurs.

Remarks

If the archive contains directories, the directory structure will be recreated in the specified directory. The [ZipItem.Name](#) property value becomes the file name of the extracted file. If a file with the same name already exists, a file name conflict occurs. The [AllowFileOverwriteMode](#) parameter specifies the default action in this situation.

 Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=E4695 .	

This example illustrates how to handle a file name conflict when files are extracted from archive. If a file with the same name exists, the [AllowFileOverwrite](#) event occurs. You can handle this event and determine that if the file in the folder is newer than the zip item, the file in the folder should not be overwritten.

C#	
-----------	--

```
(ZipExamples.cs)
using DevExpress.Compression;

public void UnzipArchiveConflict() {
    string pathToZipArchive = "Documents\\Example.zip";
    string pathToExtract = "Documents\\!Extracted";
    using (ZipArchive archive = ZipArchive.Read(pathToZipArchive)) {
        archive.OptionsBehavior.AllowFileOverwrite = AllowFileOverwriteMode.Custom;
        archive.AllowFileOverwrite += archive_AllowFileOverwrite;
        foreach (ZipItem item in archive) {
            item.Extract(pathToExtract);
        }
    }
}

private void archive_AllowFileOverwrite(object sender, AllowFileOverwriteEventArgs e) {
    FileInfo fi = new FileInfo(e.TargetFilePath);
    if (e.ZipItem.LastWriteTime < fi.LastWriteTime) e.Cancel = true;
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub UnzipArchiveConflict()
    Dim pathToZipArchive As String = "Documents\Example.zip"
    Dim pathToExtract As String = "Documents\!Extracted"
    Using archive As ZipArchive = ZipArchive.Read(pathToZipArchive)
        archive.OptionsBehavior.AllowFileOverwrite = AllowFileOverwriteMode.Custom
        AddHandler archive.AllowFileOverwrite, AddressOf archive_AllowFileOverwrite
        For Each item As ZipItem In archive
            item.Extract(pathToExtract)
        Next item
    End Using
End Sub

Private Sub archive_AllowFileOverwrite(ByVal sender As Object, ByVal e As AllowFileOverwriteEventArgs)
    Dim fi As New FileInfo(e.TargetFilePath)
    If e.ZipItem.LastWriteTime < fi.LastWriteTime Then
        e.Cancel = True
    End If
End Sub

End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.Extract Overload List](#)

IsValid Method

Reserved for future use.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function IsValid() As Boolean
```

C#

```
public bool IsValid()
```

Return Value

Always **true**.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

Read Method

Static method that creates a ZipArchive instance from the specified archive file.

Overload List

Name	Description
static ZipArchive Read(string fileName)	Static method that creates a ZipArchive instance from the specified archive file.
static ZipArchive Read(Stream stream)	Static method that creates a ZipArchive instance from the specified stream containing zipped data.
static ZipArchive Read(string fileName, Encoding encoding)	Static method that creates a ZipArchive instance from the specified archive file.
static ZipArchive Read(Stream stream, Encoding encoding)	Static method that creates a ZipArchive instance from the specified stream containing zipped data.
static ZipArchive Read(string fileName, Encoding encoding, bool catchExceptions)	Static method that creates a ZipArchive instance from the archive file, uses the specified encoding for the zip item names and allows you not to catch exceptions when extracting a particular zip entry.
static ZipArchive Read(Stream stream, Encoding encoding, bool catchExceptions)	Static method that creates a ZipArchive instance from the specified stream containing zipped data, use the specified encoding for the zip item names and allows you not to catch exceptions when extracting a particular zip entry.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.Read Overload List](#)

Static method that creates a ZipArchive instance from the specified archive file.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Read(  
    ByVal fileName As String  
) As ZipArchive
```

C#

```
public static ZipArchive Read(  
    string fileName  
)
```

Parameters

fileName
A string that is the path to the archive file.

Return Value

A [ZipArchive](#) instance that is the zip archive for modification or extraction.

Remarks

Use the **Read** method to open the archive for modification or extraction. To save the archive, use the corresponding [Save](#) method override.

Example

 Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

When the file archive is opened with the [Read](#) method, you cannot save it to the same file. The [Save](#) method attempts to overwrite the locked file resulting in an exception.

This code snippet illustrates how you can add a file to an archive and save it with the same name as before.

C#

```
(ZipExamples.cs)
public void AddFileToArchive() {
    MemoryStream stream = new MemoryStream();
    string[] sourcefiles = this.sourceFiles;
    string pathToZipArchive = "Documents\\Example.zip";
    using (FileStream fs = File.Open(pathToZipArchive, FileMode.Open)) {
        fs.CopyTo(stream);
        fs.Close();
    }
    stream.Seek(0, SeekOrigin.Begin);
    using (ZipArchive archive = ZipArchive.Read(stream, System.Text.Encoding.Default, false)) {
        foreach (string sfile in sourcefiles) {
            archive.AddFile(sfile, "/");
        }
        archive.Save(pathToZipArchive);
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Public Sub AddFileToArchive()
    Dim stream As New MemoryStream()
    Dim sourcefiles() As String = Me.sourceFiles
    Dim pathToZipArchive As String = "Documents\\Example.zip"
    Using fs As FileStream = File.Open(pathToZipArchive, FileMode.Open)
        fs.CopyTo(stream)
        fs.Close()
    End Using
    stream.Seek(0, SeekOrigin.Begin)
    Using archive As ZipArchive = ZipArchive.Read(stream, System.Text.Encoding.Default, False)
        For Each sfile As String In sourcefiles
            archive.AddFile(sfile, "/")
        Next sfile
        archive.Save(pathToZipArchive)
    End Using
End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.Read Overload List](#)

Static method that creates a ZipArchive instance from the specified stream containing zipped data.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Read(  
    ByVal stream As Stream  
) As ZipArchive  
C#  
public static ZipArchive Read(  
    Stream stream  
)
```

Parameters
stream
A [System.IO.Stream](#) object that is the stream containing archive data.

Return Value
A [ZipArchive](#) instance that is the zip archive for modification or extraction.

Remarks
Use the **Read** method to open the archive for modification or extraction. To save the archive, use the corresponding [Save](#) method override.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.Read Overload List](#)

Static method that creates a ZipArchive instance from the specified archive file.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Read(  
    ByVal fileName As String,  
    ByVal encoding As Encoding  
) As ZipArchive  
C#  
public static ZipArchive Read(  
    string fileName,  
    Encoding encoding  
)
```

Parameters
fileName
A string that is the path to the archive file.
encoding
The [System.Text.Encoding](#) object that specifies character encoding for the zip item names.

Return Value
A [ZipArchive](#) instance that is the zip archive for modification or extraction.

Remarks
Use the **Read** method to open the archive for modification or extraction. To save the archive, use the corresponding [Save](#) method override.
The **Read** method allows you to specify character encoding to correctly read item names.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=E4695>.

When the file archive is opened with the [Read](#) method, you cannot save it to the same file. The [Save](#) method attempts to overwrite the locked file resulting in an exception.

This code snippet illustrates how you can add a file to an archive and save it with the same name as before.

```
C#

(ZipExamples.cs)
public void AddFileToArchive() {
    MemoryStream stream = new MemoryStream();
    string[] sourcefiles = this.sourceFiles;
    string pathToZipArchive = "Documents\\Example.zip";
    using (FileStream fs = File.Open(pathToZipArchive, FileMode.Open)) {
        fs.CopyTo(stream);
        fs.Close();
    }
    stream.Seek(0, SeekOrigin.Begin);
    using (ZipArchive archive = ZipArchive.Read(stream, System.Text.Encoding.Default, false)) {
        foreach (string sfile in sourcefiles) {
            archive.AddFile(sfile, "/");
        }
        archive.Save(pathToZipArchive);
    }
}
```

```
Visual Basic

(ZipExamples.vb)
Public Sub AddFileToArchive()
    Dim stream As New MemoryStream()
    Dim sourcefiles() As String = Me.sourceFiles
    Dim pathToZipArchive As String = "Documents\\Example.zip"
    Using fs As FileStream = File.Open(pathToZipArchive, FileMode.Open)
        fs.CopyTo(stream)
        fs.Close()
    End Using
    stream.Seek(0, SeekOrigin.Begin)
    Using archive As ZipArchive = ZipArchive.Read(stream, System.Text.Encoding.Default, False)
        For Each sfile As String In sourcefiles
            archive.AddFile(sfile, "/")
        Next sfile
        archive.Save(pathToZipArchive)
    End Using
End Sub
```

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.Read Overload List](#)

Static method that creates a ZipArchive instance from the specified stream containing zipped data.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Read(
    ByVal stream As Stream,
    ByVal encoding As Encoding
) As ZipArchive
```

C#

```
public static ZipArchive Read(  
    Stream stream,  
    Encoding encoding  
)
```

Parameters

stream

A [System.IO.Stream](#) object that is the stream containing archive data.

encoding

The [System.Text.Encoding](#) object that specifies character encoding for the zip item names.

Return Value

A [ZipArchive](#) instance that is the zip archive for modification or extraction.

Remarks

Use the **Read** method to open the archive for modification or extraction. To save the archive, use the corresponding [Save](#) method override.

The **Read** method allows you to specify character encoding to correctly read item names.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.Read Overload List](#)

Static method that creates a ZipArchive instance from the archive file, uses the specified encoding for the zip item names and allows you not to catch exceptions when extracting a particular zip entry.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Read(  
    ByVal fileName As String,  
    ByVal encoding As Encoding,  
    ByVal catchExceptions As Boolean  
) As ZipArchive
```

C#

```
public static ZipArchive Read(  
    string fileName,  
    Encoding encoding,  
    bool catchExceptions  
)
```

Parameters

fileName

A string that is the path to the archive file.

encoding

The [System.Text.Encoding](#) object that specifies character encoding for the zip item names.

catchExceptions

True, to catch exceptions when extracting a particular zip entry and skip the problematic entry; otherwise, **false**.

Return Value

A [ZipArchive](#) instance that is the zip archive for modification or extraction.

Remarks

The **Read** method override allows you not to skip problematic zip entries during archive extraction (default behavior), but to throw exceptions so they propagate up the call chain.

If you handle the [Error](#) event in a problematic situation, the event occurs and subsequently, the exception is thrown.



Note

When the file archive is opened with the [Read](#) method, you cannot save it to the same file because the original file is locked. The [Save](#) method always attempts to create a new file overwriting the existing one.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.Read Overload List](#)

Static method that creates a ZipArchive instance from the specified stream containing zipped data, use the specified encoding for the zip item names and allows you not to catch exceptions when extracting a particular zip entry.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Read(  
    ByVal stream As Stream,  
    ByVal encoding As Encoding,  
    ByVal catchExceptions As Boolean  
) As ZipArchive
```

C#

```
public static ZipArchive Read(  
    Stream stream,  
    Encoding encoding,  
    bool catchExceptions  
)
```

Parameters

stream

A [System.IO.Stream](#) object that is the stream containing archive data.

encoding

The [System.Text.Encoding](#) object that specifies character encoding for the zip item names.

catchExceptions

True to catch exceptions when extracting a particular zip entry and skip the problematic entry; otherwise, **false**.

Return Value

A [ZipArchive](#) instance that is the zip archive for modification or extraction.

Remarks

The **Read** method override allows you not to skip problematic zip entries during archive extraction (default behavior), but to throw exceptions so they propagate up the call chain.



Note

If you handle the [Error](#) event, in the problematic situation the event occurs and subsequently the exception is thrown.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.Read Overload List](#)

RemoveItem Method

Deletes a specified zip item from the archive.

Overload List

Name	Description
void RemoveItem(ZipItem item)	Deletes a specified zip item from the archive.
void RemoveItem(string name)	Deletes a specified zip item from the archive.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.RemoveItem Overload List](#)

Deletes a specified zip item from the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub RemoveItem(  
    ByVal item As ZipItem  
)
```

C#

```
public void RemoveItem(  
    ZipItem item  
)
```

Parameters

item
A [ZipItem](#) instance to remove.

Remarks

The **RemoveItem** method removes the item from the [ZipArchive](#) instance, but does not update the archive file or stream. Call the [Save](#) method to save the modified archive.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.RemoveItem Overload List](#)

Deletes a specified zip item from the archive.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub RemoveItem(  
    ByVal name As String  
)
```

C#

```
public void RemoveItem(  
    string name  
)
```

Parameters

name

A string that is the name of a zip item to remove.

Remarks

The name of the zip item corresponds to the [ZipItem.Name](#) property value. The **RemoveItem** method removes the item from the [ZipArchive](#) instance, but does not update the archive file or stream. Call the [Save](#) method to save the modified archive.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.RemoveItem Overload List](#)

Save Method

Compresses data and saves it to a specified stream.

Overload List

Name	Description
void Save(Stream stream)	Compresses data and saves it to a specified stream.
void Save(string fileName)	Compresses data and saves it to a file.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.Save Overload List](#)

Compresses data and saves it to a specified stream.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Save(  
    ByVal stream As Stream  
)
```


C#

```
public void Save(  
    Stream stream  
)
```

Parameters

stream
The [System.IO.Stream](#) object to output the archive to.

Remarks

 **Note**

Do not save an archive to a stream that does not support seek operations. An exception is thrown if this happens. Use the **System.IO.Stream.CanSeek** property to determine if the steam supports seeking.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.Save Overload List](#)

Compresses data and saves it to a file.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Save(  
    ByVal fileName As String  
)
```

C#

```
public void Save(  
    string fileName  
)
```

Parameters*fileName*

A string that is the file name of the archive to store zipped data.

Remarks

Use the **Save** to store compressed data. To load an archive from the file, use the [Read](#) method.

When zip items are being archived, the [Progress](#) event occurs. It allows you to interrupt the process.

Note that not all zip item properties are stored in the file. The [ZipFileItem.FileName](#) property is discarded.

See Also[ZipArchive Class](#)[ZipArchive Members](#)[DevExpress.Compression Namespace](#)[ZipArchive.Save Overload List](#)

UpdateDirectory Method

Recursively updates files and nested directories starting from the specified directory.

Overload List

Name	Description
ZipDirectoryItem UpdateDirectory(string path)	Recursively updates files and nested directories starting from the specified directory.
ZipDirectoryItem UpdateDirectory(string path, string archivePath)	Recursively updates files and nested directories starting from the specified directory.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.UpdateDirectory Overload List](#)

Recursively updates files and nested directories starting from the specified directory.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function UpdateDirectory(  
    ByVal path As String  
) As ZipDirectoryItem
```

C#

```
public ZipDirectoryItem UpdateDirectory(  
    string path  
)
```

Parameters

path
A string that is the path to a directory from which the update starts.

Return Value

Currently, always returns **null**.

Remarks

The **UpdateDirectory** method recursively traverses all directories starting from the specified path. For each file, it calls the [UpdateFile](#) method so that the archive path equals the file path without the root component. As a result, all file items in the archive which correspond to files in the file system are replaced with new items created from files. If a zip item that corresponds to the file is not found, it is created and added to the archive. All file items which do not correspond to files are left intact.

Empty directories are skipped and not created in the archive.

Use the [Read](#) to load the archive to be updated. To save the updated archive, use the [Save](#) method.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.UpdateDirectory Overload List](#)

Recursively updates files and nested directories starting from the specified directory.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function UpdateDirectory(  
    ByVal path As String,  
    ByVal archivePath As String  
) As ZipDirectoryItem
```

C#

```
public ZipDirectoryItem UpdateDirectory(  
    string path,  
    string archivePath  
)
```

Parameters

path

A string that is the path to a directory from which the update starts.

archivePath

A string that is the path in the archive.

Return Value

Currently, always returns **null**.

Remarks

The **UpdateDirectory** method recursively traverses all directories starting from the specified path. For each file, it calls the [UpdateFile](#) method. As a result, all file items in the archive which correspond to files in the file system are replaced with new items created from files. If an item that corresponds to the file is not found, it is created and added to the archive. All file items which do not correspond to files are left intact.

Empty directories are skipped and not created in the archive.

Use the [Read](#) to load the archive to be updated. To save the updated archive, use the [Save](#) method.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.UpdateDirectory Overload List](#)

UpdateFile Method

Updates content of the zip file item.

Overload List

Name	Description
ZipFileItem.UpdateFile(string fileName)	Updates content of the zip file item.
ZipFileItem.UpdateFile(string fileName, string relativePath)	Updates content of the zip file item.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.UpdateFile Overload List](#)

Updates content of the zip file item.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function UpdateFile(  
    ByVal fileName As String  
) As ZipFileItem
```

C#

```
public ZipFileItem UpdateFile(  
    string fileName  
)
```

Parameters

fileName
A string that is the path to the file which will replace the file in the archive.

Return Value

A [ZipFileItem](#) object that is the updated zip file item.

Remarks

The **UpdateFile** method removes a file item located at the path which equals the path to the file without the root component. Then, the [ZipArchive](#) creates a new zip file item from the specified file and adds it to the archive at the same path.

If an item is not found, it is created.

Use the [Read](#) to load the archive to be updated. To save the updated archive, use the [Save](#) method.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.UpdateFile Overload List](#)

Updates content of the zip file item.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function UpdateFile(  
    ByVal fileName As String,  
    ByVal relativePath As String  
) As ZipFileItem
```

C#

```
public ZipFileItem UpdateFile(  
    string fileName,  
    string relativePath  
)
```

Parameters*fileName*

A string that is the path to the file which will replace the file in the archive.

relativePath

A string that is the path in archive.

Return Value

A [ZipFileItem](#) object that is the updated zip file item.

Remarks

The **UpdateFile** method removes a file item located at the specified path from the archive. Then, the [ZipArchive](#) creates a new zip file item from the specified file and adds it to the archive at the specified path.

If an item is not found, it is created.

Use the [Read](#) to load the archive to be updated. To save the updated archive, use the [Save](#) method.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.UpdateFile Overload List](#)

UpdateStream Method

Updates the content of the zip stream item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function UpdateStream(  
    ByVal name As String,  
    ByVal stream As Stream  
) As ZipStreamItem
```

C#

```
public ZipStreamItem UpdateStream(  
    string name,  
    Stream stream  
)
```

Parameters

name

A string that is the name of the zip stream item to update.

stream

A [System.IO.Stream](#) object that is the new stream of data for the zip stream item.

Return Value

A [ZipStreamItem](#) object that is the updated zip stream item.

Remarks

The **UpdateStream** method removes an item from the archive, creates a new zip stream item with the same name along with a new stream and adds it to the archive.

If an item is not found, it is created.

Use the [Read](#) to load the archive to be updated. To save the updated archive, use the [Save](#) method.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

UpdateText Method

Updates the content of the zip text item.

Overload List

Name	Description
ZipTextItem UpdateText(string name, string content)	Updates the content of the zip text item.
ZipTextItem UpdateText(string name, string content, Encoding encoding)	Updates the content of the zip text item and specifies a new character encoding.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.UpdateText Overload List](#)

Updates the content of the zip text item.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function UpdateText(  
    ByVal name As String,  
    ByVal content As String  
) As ZipTextItem
```

C#

```
public ZipTextItem UpdateText(  
    string name,  
    string content  
)
```

Parameters

name
A string that is the name of the zip text item to update.
content
A string that is the new text content of the zip text item.

Return Value

A [ZipTextItem](#) object that is the updated zip text item.

Remarks

The **UpdateText** method removes an item from the archive, creates a new zip text item with the same name and the new content and adds it to the archive.

If an item is not found, it is created.

Use the [Read](#) to load the archive to be updated. To save the updated archive, use the [Save](#) method.

See Also
[ZipArchive Class](#)
[ZipArchive Members](#)
[DevExpress.Compression Namespace](#)
[ZipArchive.UpdateText Overload List](#)

Updates the content of the zip text item and specifies a new character encoding.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function UpdateText(  
    ByVal name As String,  
    ByVal content As String,  
    ByVal encoding As Encoding  
) As ZipTextItem
```

C#

```
public ZipTextItem UpdateText(  
    string name,  
    string content,  
    Encoding encoding  
)
```

Parameters

name

A string that is the name of the zip text item to update.

content

A string that is the new text content of the zip text item.

encoding

A [System.Text.Encoding](#) object that specifies the character encoding of the text content.

Return Value

A [ZipTextItem](#) object that is the updated zip text item.

Remarks

The **UpdateText** method removes an item from the archive, creates a new zip text item with the same name, new content and character encoding, and adds it to the archive.

If an item is not found, it is created.

Use the [Read](#) to load the archive to be updated. To save the updated archive, use the [Save](#) method.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

[ZipArchive.UpdateText Overload List](#)

Validate Method

Verifies archive entries by extracting them into memory and catching exceptions.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Validate() As List(of ZipItem)
```

C#

```
public List<ZipItem> Validate()
```

Return Value

A `System.Collections.Generic.List<DevExpress.Compression.ZipItem>` list of archive items which cannot be successfully extracted.

Remarks

The **Validate** method allows you to verify an archive and obtain a list of problematic [ZipItem](#) elements.

See Also

[ZipArchive Class](#)

[ZipArchive Members](#)

[DevExpress.Compression Namespace](#)

ZipArchiveOptionsBehavior Class

Contains options that specify how the zip archive performs certain actions.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class ZipArchiveOptionsBehavior Inherits CompressionNotificationOptions
```

C#

```
public class ZipArchiveOptionsBehavior : CompressionNotificationOptions
```

Inheritance Hierarchy

[Object](#)

ViewStatePersisterCore

BaseOptions

DevExpress.Compression.CompressionNotificationOptions

ZipArchiveOptionsBehavior

See Also

[ZipArchiveOptionsBehavior Members](#)


[DevExpress.Compression Namespace](#)

ZipArchiveOptionsBehavior Members


Contains options that specify how the zip archive performs certain actions.

The following tables list the members exposed by the [ZipArchiveOptionsBehavior](#) type.








Public Constructors






	Name	Description
	ZipArchiveOptionsBehavior	Initializes a new instance of the ZipArchiveOptionsBehavior class with default settings.

Public Properties

	Name	Description
	AllowFileOverwrite	Gets or sets the default behavior in case of a file conflict when extracting files from an archive.

Public Methods

	Name	Description
	Assign	Copies all settings from the options object passed as a parameter. (Inherited from BaseOptions)
	BeginUpdate	Locks the BaseOptions object by disallowing visual updates until the EndUpdate or CancelUpdate method is called. (Inherited from BaseOptions)
	CancelUpdate	Unlocks the BaseOptions object after it has been locked by the BeginUpdate method, without causing an immediate visual update. (Inherited from BaseOptions)
	EndUpdate	Unlocks the BaseOptions object after a call to the BeginUpdate method and causes an immediate visual update. (Inherited from BaseOptions)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)

	GetType	Gets the System.Type of the current instance. (Inherited from Object)
 	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	Reset	(Inherited from DevExpress.Compression.CompressionNotificationOptions)
	ToString	Returns a string representing the currently enabled options. (Inherited from BaseOptions)

See Also
[ZipArchiveOptionsBehavior Members](#)
[DevExpress.Compression Namespace](#)

ZipArchiveOptionsBehavior Constructor

Initializes a new instance of the [ZipArchiveOptionsBehavior](#) class with default settings.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public ZipArchiveOptionsBehavior()
```

See Also

[ZipArchiveOptionsBehavior Class](#)

[ZipArchiveOptionsBehavior Members](#)


[DevExpress.Compression Namespace](#)

ZipArchiveOptionsBehavior Properties


Contains options that specify how the zip archive performs certain actions.

The following tables list the members exposed by the [ZipArchiveOptionsBehavior](#) type.








Public Constructors






	Name	Description
	ZipArchiveOptionsBehavior	Initializes a new instance of the ZipArchiveOptionsBehavior class with default settings.

Public Properties

	Name	Description
	AllowFileOverwrite	Gets or sets the default behavior in case of a file conflict when extracting files from an archive.

Public Methods

	Name	Description
	Assign	Copies all settings from the options object passed as a parameter. (Inherited from BaseOptions)
	BeginUpdate	Locks the BaseOptions object by disallowing visual updates until the EndUpdate or CancelUpdate method is called. (Inherited from BaseOptions)
	CancelUpdate	Unlocks the BaseOptions object after it has been locked by the BeginUpdate method, without causing an immediate visual update. (Inherited from BaseOptions)
	EndUpdate	Unlocks the BaseOptions object after a call to the BeginUpdate method and causes an immediate visual update. (Inherited from BaseOptions)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)

	GetType	Gets the System.Type of the current instance. (Inherited from Object)
 	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	Reset	(Inherited from DevExpress.Compression.CompressionNotificationOptions)
	ToString	Returns a string representing the currently enabled options. (Inherited from BaseOptions)

See Also
[ZipArchiveOptionsBehavior Members](#)
[DevExpress.Compression Namespace](#)

AllowFileOverwrite Property

Gets or sets the default behavior in case of a file conflict when extracting files from an archive.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property AllowFileOverwrite As AllowFileOverwriteMode
```

C#

```
public AllowFileOverwriteMode AllowFileOverwrite { get; set; }
```

Property Value

An [AllowFileOverwriteMode](#) enumeration value that specifies the default behavior.

Remarks

You can decide whether the file with the same name should be overwritten for each file individually by setting the **AllowFileOverwrite** to [AllowFileOverwriteMode.Custom](#) and handling the [ZipArchive.AllowFileOverwrite](#) event.

See Also

[ZipArchiveOptionsBehavior Class](#)

[ZipArchiveOptionsBehavior Members](#)

[DevExpress.Compression Namespace](#)

ZipByteArrayItem Class

A zip item specific to the byte array source.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class ZipByteArrayItem Inherits ZipItem
```

C#

```
public class ZipByteArrayItem : ZipItem
```

Remarks

The **ZipByteArrayItem** instance is obtained by using the [ZipArchive.AddByteArray](#) method to add a new zip item from the specified array of bytes.

Inheritance Hierarchy

[Object](#)

[ZipItem](#)

ZipByteArrayItem

See Also

[ZipByteArrayItem Members](#)

[DevExpress.Compression Namespace](#)

ZipByteArrayItem Members






A zip item specific to the byte array source.

The following tables list the members exposed by the [ZipByteArrayItem](#) type.

Public Constructors









	Name	Description
	ZipByteArrayItem	Initializes a new instance of the ZipByteArrayItem class with the specified name and content.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	Content	Gets or sets the content of the zip item created from the byte array.
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)
	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)

	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)
--	----------------------------------	--

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Extract	Overloaded. Extracts the current archive item as a file into the specified directory and enables you to define the behavior in case of a file name conflict. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[ZipByteArrayItem Members](#)
[DevExpress.Compression Namespace](#)

ZipByteArrayItem Constructor

Initializes a new instance of the [ZipByteArrayItem](#) class with the specified name and content.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal name As String,  
    ByVal content As Byte[]  
)
```

C#

```
public ZipByteArrayItem(  
    string name,  
    Byte[] content  
)
```

Parameters

name

A string that specifies the name of the zip item.

content

A System.Byte[] array to compress and store in a zip item.

See Also

[ZipByteArrayItem Class](#)

[ZipByteArrayItem Members](#)

[DevExpress.Compression Namespace](#)

ZipByteArrayItem Properties



A zip item specific to the byte array source.

The following tables list the members exposed by the [ZipByteArrayItem](#) type.

Public Constructors









	Name	Description
	ZipByteArrayItem	Initializes a new instance of the ZipByteArrayItem class with the specified name and content.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	Content	Gets or sets the content of the zip item created from the byte array.
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)
	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)

	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)
--	----------------------------------	--

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Extract	Overloaded. Extracts the current archive item as a file into the specified directory and enables you to define the behavior in case of a file name conflict. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[ZipByteArrayItem Members](#)
[DevExpress.Compression Namespace](#)

Content Property

Gets or sets the content of the zip item created from the byte array.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Content As Byte[]
```

C#

```
public Byte[] Content { get; set; }
```

Property Value

A System.Byte[] array of bytes that is the data for archiving.

See Also

[ZipByteArrayItem Class](#)

[ZipByteArrayItem Members](#)

[DevExpress.Compression Namespace](#)

ZipDirectoryItem Class

A zip item specific to the directory.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class ZipDirectoryItem Inherits ZipItem
```

C#

```
public class ZipDirectoryItem : ZipItem
```

Inheritance Hierarchy

[Object](#)

[ZipItem](#)

ZipDirectoryItem

See Also

[ZipDirectoryItem Members](#)

[DevExpress.Compression Namespace](#)

ZipDirectoryItem Members












A zip item specific to the directory.

The following tables list the members exposed by the [ZipDirectoryItem](#) type.

Public Constructors

	Name	Description
	ZipDirectoryItem	Initializes a new instance of the ZipDirectoryItem class with the specified directory name.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	DirectoryName	Gets or sets the name of a directory for the zip directory item.
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)
	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)

	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)
--	----------------------------------	--

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts archive item as a file to the specified directory. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipDirectoryItem Members](#)[DevExpress.Compression Namespace](#)

ZipDirectoryItem Constructor

Initializes a new instance of the [ZipDirectoryItem](#) class with the specified directory name.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal name As String,  
    ByVal dirName As String  
)
```

C#

```
public ZipDirectoryItem(  
    string name,  
    string dirName  
)
```

Parameters

name

A string that specifies the name of the zip item.

dirName

A string that specifies the name of the directory.

See Also

[ZipDirectoryItem Class](#)

[ZipDirectoryItem Members](#)

[DevExpress.Compression Namespace](#)

ZipDirectoryItem Properties










A zip item specific to the directory.

The following tables list the members exposed by the [ZipDirectoryItem](#) type.

Public Constructors

	Name	Description
	ZipDirectoryItem	Initializes a new instance of the ZipDirectoryItem class with the specified directory name.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	DirectoryName	Gets or sets the name of a directory for the zip directory item.
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)
	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)

	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)
--	----------------------------------	--

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts archive item as a file to the specified directory. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipDirectoryItem Members](#)[DevExpress.Compression Namespace](#)

DirectoryName Property

Gets or sets the name of a directory for the zip directory item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property DirectoryName As String
```

C#

```
public string DirectoryName { get; set; }
```

Property Value

A string that is the directory name.

See Also

[ZipDirectoryItem Class](#)

[ZipDirectoryItem Members](#)

[DevExpress.Compression Namespace](#)

ZipFileItem Class

A zip item specific to the file source.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class ZipFileItem Inherits [ZipItem](#)

C#

public class ZipFileItem : [ZipItem](#)

Remarks

The **ZipFileItem** instance is obtained by using the [ZipArchive.AddFile](#) method to add a new zip item from the file source.

The zip file item has a [FileName](#) property that is the file path of the original file and the [ZipItem.LastAccessTimeUtc](#), [ZipItem.LastWriteTimeUtc](#) and [ZipItem.CreationTimeUtc](#) properties with values taken from the corresponding attributes of the original file.

Inheritance Hierarchy

[Object](#)

[ZipItem](#)

ZipFileItem

See Also

[ZipFileItem Members](#)


[DevExpress.Compression Namespace](#)

ZipFileItem Members









A zip item specific to the file source.

The following tables list the members exposed by the [ZipFileItem](#) type.

Public Constructors

	Name	Description
	ZipFileItem	Initializes a new instance of the ZipFileItem class with the specified name and file name.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	FileName	Gets or sets the name of the file from which a zip item is created.
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)
	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)

	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)
--	----------------------------------	--

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts archive item as a file to the specified directory. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipFileItem Members](#)[DevExpress.Compression Namespace](#)

ZipFileItem Constructor

Initializes a new instance of the [ZipFileItem](#) class with the specified name and file name.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal name As String,  
    ByVal fileName As String  
)
```

C#

```
public ZipFileItem(  
    string name,  
    string fileName  
)
```

Parameters

name

A string that specifies the name of the zip item.

fileName

A string that is the path to the file being archived.

See Also

[ZipFileItem Class](#)

[ZipFileItem Members](#)


[DevExpress.Compression Namespace](#)

ZipFileItem Properties












A zip item specific to the file source.

The following tables list the members exposed by the [ZipFileItem](#) type.

Public Constructors

	Name	Description
	ZipFileItem	Initializes a new instance of the ZipFileItem class with the specified name and file name.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	FileName	Gets or sets the name of the file from which a zip item is created.
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)
	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)

	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)
--	----------------------------------	--

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts archive item as a file to the specified directory. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipFileItem Members](#)[DevExpress.Compression Namespace](#)

FileName Property

Gets or sets the name of the file from which a zip item is created.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property FileName As String
```

C#

```
public string FileName { get; }
```

Property Value

A string that is the file name.

Remarks

The **FileName** property contains the path to the file from which a zip item has been created. This information is not stored in an archive file when it is saved using the [ZipArchive.Save](#) method.

The **FileName** property is only relevant for the [ZipFileItem](#) instance. For zip items created from sources other than files, the **FileName** is **null**.

See Also

[ZipFileItem Class](#)

[ZipFileItem Members](#)

[DevExpress.Compression Namespace](#)

ZipItem Class

An entry in the zip archive containing compressed data.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class ZipItem Inherits Object, IContentStreamOwner
```

C#

```
public abstract class ZipItem : object, IContentStreamOwner
```

Remarks

The **ZipItem** descendants form the [ZipArchive](#). Each zip item is accessible in the archive by its index or its [Name](#) via the [ZipArchive.Item](#) property.

When a new archive is created, a zip item is created from the source data and contains information on its source and source specifics (e.g., the [ZipFileItem.FileName](#) and the [LastAccessTimeUtc](#) properties for the [ZipFileItem](#), and the [ZipTextItem.ContentEncoding](#) property for the [ZipTextItem](#)). When an archive is loaded from a file or a stream using the [ZipArchive.Read](#) method, all zip items contain only the basic information relevant to the **ZipItem** class.

An item in an archive is located at the path specified by the [Name](#) property.

To extract an item, use the [Extract](#) method.

To update an item, remove the item from the archive and add a new item with the same name. To help you in doing so, the following methods are implemented: [ZipArchive.UpdateDirectory](#), [ZipArchive.UpdateFile](#), [ZipArchive.UpdateStream](#) and [ZipArchive.UpdateText](#).

An item content can be encrypted with a password. Use the [EncryptionType](#) and the [Password](#) properties to set the encryption.

Inheritance Hierarchy

[Object](#)

ZipItem

Derived classes

See Also

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)




[Examples](#)

ZipItem Members


An entry in the zip archive containing compressed data.








The following tables list the members exposed by the [ZipItem](#) type.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory.
	Comment	Gets or sets a text comment for the zip item.
	CompressedSize	Obtains the size of compressed data in a zip item.
	CreationTime	Gets or sets the creation time for the zip item.
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item.
	Encoding	Gets or sets the character encoding for the zip item.
	EncryptionType	Gets or sets the encryption type for the zip item.
	LastAccessTime	Gets or sets the last access time for the zip item.
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item.
	LastWriteTime	Gets or sets the last write time for the zip item.
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item.
	Name	Gets or sets the name of the zip item.
	Password	Gets or sets a password used for encrypting the content of a zip item.
	UncompressedSize	Obtains the size of uncompressed data in the zip item.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)

	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts an item from the archive to the current directory.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)


See Also[ZipItem Members](#)[DevExpress.Compression Namespace](#)[Examples](#)

ZipItem Properties


An entry in the zip archive containing compressed data.








The following tables list the members exposed by the [ZipItem](#) type.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory.
	Comment	Gets or sets a text comment for the zip item.
	CompressedSize	Obtains the size of compressed data in a zip item.
	CreationTime	Gets or sets the creation time for the zip item.
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item.
	Encoding	Gets or sets the character encoding for the zip item.
	EncryptionType	Gets or sets the encryption type for the zip item.
	LastAccessTime	Gets or sets the last access time for the zip item.
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item.
	LastWriteTime	Gets or sets the last write time for the zip item.
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item.
	Name	Gets or sets the name of the zip item.
	Password	Gets or sets a password used for encrypting the content of a zip item.
	UncompressedSize	Obtains the size of uncompressed data in the zip item.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)

	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts an item from the archive to the current directory.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipItem Members](#)[DevExpress.Compression Namespace](#)[Examples](#)

Attributes Property

Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Attributes As FileAttributes
```

C#

```
public FileAttributes Attributes { get; set; }
```

Property Value

A System.IO.FileAttributes enumeration that specifies attributes for files or directories.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

Comment Property

Gets or sets a text comment for the zip item.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property Comment As [String](#)

C#

public [string](#) Comment { get; set; }

Property Value

A string that is stored in an archive as a comment to the zip item.

Remarks

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example adds a text comment to each zip item in the archive that indicates the person who is currently logged on.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void ArchiveWithComment() {
    string path = this.startupPath;
    using (ZipArchive archive = new ZipArchive()) {
        foreach (string file in System.IO.Directory.EnumerateFiles(path)) {
            ZipFileItem zipFI = archive.AddFile(file, "/");
            zipFI.Comment = "Archived by " + Environment.UserName;
        }
        archive.Save("Documents\\ArchiveWithComment.zip");
    }
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub ArchiveWithComment()
    Dim path As String = Me.startupPath
    Using archive As New ZipArchive()
        For Each file As String In System.IO.Directory.EnumerateFiles(path)
            Dim zipFI As ZipFileItem = archive.AddFile(file, "/")
            zipFI.Comment = "Archived by " & Environment.UserName
        Next file
        archive.Save("Documents\\ArchiveWithComment.zip")
    End Using
End Sub
```

See Also

- [ZipItem Class](#)
- [ZipItem Members](#)
- [DevExpress.Compression Namespace](#)

CompressedSize Property

Obtains the size of compressed data in a zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property CompressedSize As Int64
```

C#

```
public Int64 CompressedSize { get; }
```

Property Value

An integer that is the size of data in bytes.

Remarks

For the newly added item, the **CompressedSize** equals to 0, since the data is being compressed when the archive is saved. To get the compressed size for the zip item, load an archive using the [ZipArchive.Read](#) method.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

CreationTime Property

Gets or sets the creation time for the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CreationTime As DateTime
```

C#

```
public DateTime CreationTime { get; set; }
```

Property Value

A [System.DateTime](#) value that is the creation time.

Remarks

The **CreationTime** property value is calculated from the [CreationTimeUtc](#) property value according to the local time of the host.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

CreationTimeUtc Property

Gets or sets the coordinated universal time value of the creation time for the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property CreationTimeUtc As DateTime
```

C#

```
public DateTime CreationTimeUtc { get; set; }
```

Property Value

A [System.DateTime](#) value that is the creation time value expressed in coordinated universal time (UTC).

Remarks

The [ZipArchive.AddFile](#) and [ZipArchive.AddDirectory](#) methods set the **CreationTimeUtc** to the creation time in UTC of the file or directory being zipped. The [ZipArchive.AddByteArray](#), [ZipArchive.AddText](#) and [ZipArchive.AddStream](#) methods set the **CreationTimeUtc** to the current time in UTC.

The **CreationTimeUtc** value is stored in an archive file when it is saved using the [ZipArchive.Save](#) method.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

Encoding Property

Gets or sets the character encoding for the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property Encoding As [Encoding](#)

C#

```
public Encoding Encoding { get; set; }
```

Property Value

A [System.Text.Encoding](#) object that specifies character encoding. By default it is UTF-8.

Remarks

The **Encoding** value is used for proper conversion of the text content of the [ZipTextItem](#) into a stream for archiving.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

EncryptionType Property

Gets or sets the encryption type for the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Overridable Public Property EncryptionType As [EncryptionType](#)

C#

```
public virtual EncryptionType EncryptionType { get; set; }
```

Property Value

An [EncryptionType](#) enumeration member specifying the encryption type. Default is [EncryptionType.None](#).

Remarks

When a password is set for the zip item, the encryption type is automatically set to [EncryptionType.PkZip](#).

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

LastAccessTime Property

Gets or sets the last access time for the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property LastAccessTime As DateTime
```

C#

```
public DateTime LastAccessTime { get; set; }
```

Property Value

A [System.DateTime](#) value that is the last access time.

Remarks

The **LastAccessTime** property value is calculated from the [LastAccessTimeUtc](#) property value according to the local time of the host.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

LastAccessTimeUtc Property

Gets or sets the coordinated universal time value for the last access time for the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property LastAccessTimeUtc As DateTime
```

C#

```
public DateTime LastAccessTimeUtc { get; set; }
```

Property Value

A [System.DateTime](#) value that is the coordinated universal time (UTC) value for the last access time.

Remarks

The **LastAccessTimeUtc** property value is set to the last access time of the source file if the [ZipFileItem](#) is created from the file or directory. For the [ZipTextItem](#), the [ZipStreamItem](#) and the [ZipByteArrayItem](#), the **LastAccessTimeUtc** is set by default to the time when the item was created.



Note

Different file systems record the file last access time in a different ways, with different time resolutions for updating the last access time. Therefore, you should be aware of this when comparing the last access times for files in different file systems.

The **LastAccessTimeUtc** value is stored in an archive file when it is saved using the [ZipArchive.Save](#) method.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

LastWriteTime Property

Gets or sets the last write time for the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property LastWriteTime As DateTime
```

C#

```
public DateTime LastWriteTime { get; set; }
```

Property Value

A [System.DateTime](#) value that is the last write time.

Remarks

The **LastWriteTime** property value is calculated from the [LastWriteTimeUtc](#) property value based on the local time of the host.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

LastWriteTimeUtc Property

Gets or sets the coordinated universal time value for the last write time for the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property LastWriteTimeUtc As DateTime
```

C#

```
public DateTime LastWriteTimeUtc { get; set; }
```

Property Value

A [System.DateTime](#) value that is the coordinated universal time (UTC) value for the last write time.

Remarks

The **LastWriteTimeUtc** property value is set to the last write time of the source file if the [ZipFileItem](#) is created from the file or directory. For the [ZipTextItem](#), the [ZipStreamItem](#) and the [ZipByteArrayItem](#), the **LastWriteTimeUtc** is set by default to the time when the item was created.

The **LastWriteTimeUtc** value is stored in an archive file when it is saved using the [ZipArchive.Save](#) method.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

Name Property

Gets or sets the name of the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Name As String
```

C#

```
public string Name { get; set; }
```

Property Value

A string that is the name of the zip item.

Remarks

The **Name** value is used to search for the zip item by using the [ZipArchive.Item](#) method or to remove the item by using the [ZipArchive.RemoveItem](#) method. When a zip item is extracted to a file by using the [Extract](#) or the [ZipArchive.Extract](#) method, the **Name** value is used to construct a file name.

When a [ZipArchive.AddFile](#) or the [ZipArchive.AddDirectory](#) methods are used to add a zip item to the archive, the **Name** is set to the file (directory) path without the root component. In this situation, the [ZipFileItem.FileName](#) property contains the full file path, including the root component.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

Password Property

Gets or sets a password used for encrypting the content of a zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property Password As [String](#)

C#

```
public string Password { get; set; }
```

Property Value

A string that specifies the password for encryption.

Remarks

When you specify a [Password](#) value for the zip item with the [EncryptionType](#) set to [EncryptionType.None](#) (default value), then the [EncryptionType](#) is set to [EncryptionType.PkZip](#) automatically.

The **Password** property enables you to specify a distinct password for each zip item. To set a password for the entire archive, use the [ZipArchive.Password](#) property.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

UncompressedSize Property

Obtains the size of uncompressed data in the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property UncompressedSize As Int64
```

C#

```
public Int64 UncompressedSize { get; }
```

Property Value

An integer that is the size of uncompressed data in bytes.

See Also

[ZipItem Class](#)

[ZipItem Members](#)








[DevExpress.Compression Namespace](#)

ZipItem Methods

An entry in the zip archive containing compressed data.

The following tables list the members exposed by the [ZipItem](#) type.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts an item from the archive to the current directory.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

[Examples](#)

Extract Method

Extracts an item from the archive to the current directory.

Overload List

Name	Description
void Extract()	Extracts an item from the archive to the current directory.
void Extract(Stream stream)	Extracts an item from the archive to the specified stream.
void Extract(string directory)	Extracts archive item as a file to the specified directory.
void Extract(string directory, AllowFileOverwriteMode allowFileOverwrite)	Extracts the current archive item as a file into the specified directory and enables you to define the behavior in case of a file name conflict.

See Also
[ZipItem Class](#)
[ZipItem Members](#)
[DevExpress.Compression Namespace](#)
[ZipItem.Extract Overload List](#)

Extracts an item from the archive to the current directory.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Extract()
```

C#

```
public void Extract()
```

Remarks

The path in which the zip item is located in the archive is recreated in the current directory. The current directory is the root directory for archive paths.

See Also
[ZipItem Class](#)
[ZipItem Members](#)
[DevExpress.Compression Namespace](#)
[ZipItem.Extract Overload List](#)

Extracts an item from the archive to the specified stream.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Extract(  
    ByVal stream As Stream  
)
```

C#

```
public void Extract(  
    Stream stream  
)
```

Parameters

stream
An [System.IO.Stream](#) object that is the stream to which the archive item is extracted.

Remarks

The **Extract** obtains the data stream from the zip item, copies it to the specified stream and closes the data stream. To obtain the original data stream, use the [Open](#) method.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

[ZipItem.Extract Overload List](#)

Extracts archive item as a file to the specified directory.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Extract(  
    ByVal directory As String  
)
```

C#

```
public void Extract(  
    string directory  
)
```

Parameters

directory

A string that is the path to the directory to which the files are extracted.

Remarks

If the archive contains directories, the directory structure will be recreated in the specified directory. The [Name](#) property value becomes the file name of the extracted file.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

[ZipItem.Extract Overload List](#)

Extracts the current archive item as a file into the specified directory and enables you to define the behavior in case of a file name conflict.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Extract(  
    ByVal directory As String,  
    ByVal allowFileOverwrite As AllowFileOverwriteMode  
)
```

C#

```
public void Extract(  
    string directory,  
    AllowFileOverwriteMode allowFileOverwrite  
)
```

Parameters

directory

A string that is the path to the directory to which the files are extracted.

allowFileOverwrite

An [AllowFileOverwriteMode](#) enumeration member that specifies the behavior if a file name conflict occurs.

Remarks

If the archive contains directories, the directory structure will be recreated in the specified directory. The [Name](#) property value becomes the file name of the extracted file. If a file with the same name already exists, a file name conflict occurs. In this situation, the [AllowFileOverwriteMode](#) parameter specifies the default action.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4695>.

This example illustrates how to handle a file name conflict when files are extracted from archive. If a file with the same name exists, the [ZipArchive.AllowFileOverwrite](#) event occurs. You can handle this event and determine that if the file in the folder is newer than the zip item, the file in the folder should not be overwritten.

C#

```
(ZipExamples.cs)
using DevExpress.Compression;

public void UnzipArchiveConflict() {
    string pathToZipArchive = "Documents\\Example.zip";
    string pathToExtract = "Documents\\!Extracted";
    using (ZipArchive archive = ZipArchive.Read(pathToZipArchive)) {
        archive.OptionsBehavior.AllowFileOverwrite = AllowFileOverwriteMode.Custom;
        archive.AllowFileOverwrite += archive_AllowFileOverwrite;
        foreach (ZipItem item in archive) {
            item.Extract(pathToExtract);
        }
    }
}

private void archive_AllowFileOverwrite(object sender, AllowFileOverwriteEventArgs e) {
    FileInfo fi = new FileInfo(e.TargetFilePath);
    if (e.ZipItem.LastWriteTime < fi.LastWriteTime) e.Cancel = true;
}
```

Visual Basic

```
(ZipExamples.vb)
Imports DevExpress.Compression

Public Sub UnzipArchiveConflict()
    Dim pathToZipArchive As String = "Documents\\Example.zip"
    Dim pathToExtract As String = "Documents\\!Extracted"
    Using archive As ZipArchive = ZipArchive.Read(pathToZipArchive)
        archive.OptionsBehavior.AllowFileOverwrite = AllowFileOverwriteMode.Custom
        AddHandler archive.AllowFileOverwrite, AddressOf archive_AllowFileOverwrite
        For Each item As ZipItem In archive
            item.Extract(pathToExtract)
        Next item
    End Using
End Sub

Private Sub archive_AllowFileOverwrite(ByVal sender As Object, ByVal e As AllowFileOverwriteEventArgs)
    Dim fi As New FileInfo(e.TargetFilePath)
    If e.ZipItem.LastWriteTime < fi.LastWriteTime Then
        e.Cancel = True
    End If
End Sub
```

- See Also**
- [ZipItem Class](#)
 - [ZipItem Members](#)
 - [DevExpress.Compression Namespace](#)
 - [ZipItem.Extract Overload List](#)

Open Method

Obtains the unzipped data stream of the zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Open() As Stream
```

C#

```
public Stream Open()
```

Return Value

A [System.IO.Stream](#) object that is the data stream.

Remarks

To extract the zip item to another stream, i.e., to copy the original stream to the specified stream, use the [Extract](#) method.

See Also

[ZipItem Class](#)

[ZipItem Members](#)

[DevExpress.Compression Namespace](#)

ZipItemAddingAction Enumeration

Lists a possible action when the ZipArchive.ItemAdding event is handled.

Namespace: [DevExpress.Compression](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum ZipItemAddingAction
```

C#

```
public enum ZipItemAddingAction
```

Members

Name	Description
Cancel	Cancel adding a current item to the archive and skip to the next item.
Continue	Add the current item to the archive and move to the next item. Default value.
Stop	Stops adding items to archive.

Remarks

Handle the [ZipArchive.ItemAdding](#) event to filter zip items while they are added to archive. You can also stop adding zip items when this action is required.

See Also

[DevExpress.Compression Namespace](#)

ZipItemAddingEventArgs Class

Provides data for the ZipArchive.ItemAdding event.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class ZipItemAddingEventArgs Inherits [EventArgs](#)

C#

```
public class ZipItemAddingEventArgs : EventArgs
```

Remarks

Handle the [ZipArchive.ItemAdding](#) event to filter zip items while they are added to archive. You can also stop adding zip items when this action is required.

Inheritance Hierarchy

[Object](#)

[EventArgs](#)

ZipItemAddingEventArgs

See Also

[ZipItemAddingEventArgs Members](#)


[DevExpress.Compression Namespace](#)

ZipItemAddingEventArgs Members



Provides data for the ZipArchive.ItemAdding event.

The following tables list the members exposed by the [ZipItemAddingEventArgs](#) type.







Public Constructors

	Name	Description
	ZipItemAddingEventArgs	Initializes a new instance of the ZipItemAddingEventArgs object with the specified zip item.


Public Properties

	Name	Description
	Action	Gets or sets the action required to perform after the ItemAdding event is handled.
	Item	Obtains the zip item being added to the archive.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
	Empty	Represents an event with no event data. (Inherited from EventArgs)

See Also[ZipItemAddingEventArgs Members](#)[DevExpress.Compression Namespace](#)

ZipItemAddingEventArgs Constructor

Initializes a new instance of the [ZipItemAddingEventArgs](#) object with the specified zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal item As ZipItem  
)
```

C#

```
public ZipItemAddingEventArgs(  
    ZipItem item  
)
```

Parameters

item

A [ZipItem](#) that is the zip item being added to the archive. This value is assigned to the [Item](#) property.

Remarks

Instances of the [ZipItemAddingEventArgs](#) class are automatically created, initialized and passed to the corresponding event handlers.

See Also

[ZipItemAddingEventArgs Class](#)

[ZipItemAddingEventArgs Members](#)


[DevExpress.Compression Namespace](#)

ZipItemAddingEventArgs Properties



Provides data for the `ZipArchive.ItemAdding` event.

The following tables list the members exposed by the [ZipItemAddingEventArgs](#) type.







Public Constructors

	Name	Description
	ZipItemAddingEventArgs	Initializes a new instance of the ZipItemAddingEventArgs object with the specified zip item.


Public Properties

	Name	Description
	Action	Gets or sets the action required to perform after the <code>ItemAdding</code> event is handled.
	Item	Obtains the zip item being added to the archive.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Fields

	Name	Description
	<code>Empty</code>	Represents an event with no event data. (Inherited from EventArgs)

See Also[ZipItemAddingEventArgs Members](#)[DevExpress.Compression Namespace](#)

Action Property

Gets or sets the action required to perform after the ItemAdding event is handled.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property Action As [ZipItemAddingAction](#)

C#

```
public ZipItemAddingAction Action { get; set; }
```

Property Value

A [ZipItemAddingAction](#) enumeration value specifying which action to perform.

Remarks

You can handle the [ZipArchive.ItemAdding](#) event and decide whether to skip the current item and not add it to the archive, or to cancel the entire archiving process.

See Also

[ZipItemAddingEventArgs Class](#)

[ZipItemAddingEventArgs Members](#)

[DevExpress.Compression Namespace](#)

Item Property

Obtains the zip item being added to the archive.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Item As ZipItem
```

C#

```
public ZipItem Item { get; }
```

Property Value

A [ZipItem](#) descendant being added to the archive.

See Also

[ZipItemAddingEventArgs Class](#)

[ZipItemAddingEventArgs Members](#)

[DevExpress.Compression Namespace](#)

ZipItemAddingEventHandler Delegate

A method that will handle the **ItemAdding** event of the SpreadsheetControl.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Delegate Sub ZipItemAddingEventHandler(  
    ByVal sender As Object,  
    ByVal args As ZipItemAddingEventArgs  
)
```

C#

```
public delegate void ZipItemAddingEventHandler(  
    object sender,  
    ZipItemAddingEventArgs args  
)
```

Parameters

The declaration of your event handler must have the same parameters as the **ZipItemAddingEventHandler** delegate declaration.

sender

The event source. This parameter identifies the ZipArchive which raised the event.

args

A [ZipItemAddingEventArgs](#) object which contains event data.

Remarks

When creating a [ZipItemAddingEventHandler](#) delegate, you identify the method that will handle the corresponding event. To associate an event with your event handler, add a delegate instance to this event. The event handler is called whenever the event occurs unless you remove the delegate. For more information about event handler delegates, see **Events and Delegates** in **MSDN**.

See Also

[DevExpress.Compression Namespace](#)

[ZipArchive.ItemAdding](#)

ZipStreamItem Class

A zip item specific to the stream source.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class ZipStreamItem Inherits [ZipItem](#)

C#

```
public class ZipStreamItem : ZipItem
```

Remarks

The **ZipStreamItem** instance is obtained by using the [ZipArchive.AddStream](#) method to add a new zip item from the data stream.

The stream in the zip stream item is the original data stream. It should be available until the archive is saved using the [ZipArchive.Save](#) method. When a zip stream item is disposed of, the original stream is not closed.

Inheritance Hierarchy

[Object](#)

[ZipItem](#)

ZipStreamItem

See Also

[ZipStreamItem Members](#)

[DevExpress.Compression Namespace](#)

ZipStreamItem Members


A zip item specific to the stream source.


The following tables list the members exposed by the [ZipStreamItem](#) type.

Public Constructors

	Name	Description
	ZipStreamItem	Initializes a new instance of the ZipStreamItem class with the specified name.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	ContentStream	Gets or sets the content of the zip stream item.
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)
	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)

	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)
--	----------------------------------	--

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts an item from the archive to the specified stream. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipStreamItem Members](#)[DevExpress.Compression Namespace](#)

ZipStreamItem Constructor

Initializes a new instance of the [ZipStreamItem](#) class with the specified name.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal name As String  
)
```

C#

```
public ZipStreamItem(  
    string name  
)
```

Parameters

name

A string that specifies the name of the zip item.

See Also

[ZipStreamItem Class](#)

[ZipStreamItem Members](#)

[DevExpress.Compression Namespace](#)

ZipStreamItem Properties










A zip item specific to the stream source.

The following tables list the members exposed by the [ZipStreamItem](#) type.

Public Constructors

	Name	Description
	ZipStreamItem	Initializes a new instance of the ZipStreamItem class with the specified name.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	ContentStream	Gets or sets the content of the zip stream item.
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)
	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)

	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)
--	----------------------------------	--

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Extract	Overloaded. Extracts an item from the archive to the specified stream. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipStreamItem Members](#)[DevExpress.Compression Namespace](#)

ContentStream Property

Gets or sets the content of the zip stream item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ContentStream As Stream
```

C#

```
public Stream ContentStream { get; set; }
```

Property Value

A [System.IO.Stream](#) object that is the content of a zip item.

Remarks

The **ContentStream** enables you to specify the stream to pack into the archive as the zip item. The specified stream is not closed and disposed automatically.

You can also use the [ZipArchive.AddStream](#) method to create a **ZipStreamItem** and add it to the archive.

See Also

[ZipStreamItem Class](#)

[ZipStreamItem Members](#)

[DevExpress.Compression Namespace](#)

ZipTextItem Class

A zip item specific to the text source.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class ZipTextItem Inherits ZipItem
```

C#

```
public class ZipTextItem : ZipItem
```

Remarks

The **ZipTextItem** instance is obtained by using the [ZipArchive.AddText](#) method to add a new zip item from the text string.

The zip text item has the [ContentEncoding](#) property that is the character encoding of the original text.

Inheritance Hierarchy

[Object](#)

[ZipItem](#)

ZipTextItem

See Also

[ZipTextItem Members](#)

[DevExpress.Compression Namespace](#)

ZipTextItem Members













A zip item specific to the text source.



The following tables list the members exposed by the [ZipTextItem](#) type.

Public Constructors









	Name	Description
	ZipTextItem	Initializes a new instance of the ZipTextItem class with the specified name.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	Content	Gets or sets the content of the text zip item.
	ContentEncoding	Gets or sets the character encoding of the text that is the content of a zip item.
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)

	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)
	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Extract	Overloaded. Extracts the current archive item as a file into the specified directory and enables you to define the behavior in case of a file name conflict. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipTextItem Members](#)[DevExpress.Compression Namespace](#)

ZipTextItem Constructor

Initializes a new instance of the [ZipTextItem](#) class with the specified name.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New(  
    ByVal name As String  
)
```

C#

```
public ZipTextItem(  
    string name  
)
```

Parameters

name

A string that specifies the name of the zip item.

Remarks

By default, UTF8 encoding is specified.

You can also use the [ZipArchive.AddText](#) method to create a [ZipTextItem](#) instance and add it to archive.

See Also

[ZipTextItem Class](#)

[ZipTextItem Members](#)

[DevExpress.Compression Namespace](#)

ZipTextItem Properties















A zip item specific to the text source.


The following tables list the members exposed by the [ZipTextItem](#) type.

Public Constructors









	Name	Description
	ZipTextItem	Initializes a new instance of the ZipTextItem class with the specified name.

Public Properties

	Name	Description
	Attributes	Gets or sets attributes for a ZipItem that is originated from a file or directory, or that will be decompressed into a file or directory. (Inherited from ZipItem)
	Comment	Gets or sets a text comment for the zip item. (Inherited from ZipItem)
	CompressedSize	Obtains the size of compressed data in a zip item. (Inherited from ZipItem)
	Content	Gets or sets the content of the text zip item.
	ContentEncoding	Gets or sets the character encoding of the text that is the content of a zip item.
	CreationTime	Gets or sets the creation time for the zip item. (Inherited from ZipItem)
	CreationTimeUtc	Gets or sets the coordinated universal time value of the creation time for the zip item. (Inherited from ZipItem)
	Encoding	Gets or sets the character encoding for the zip item. (Inherited from ZipItem)
	EncryptionType	Gets or sets the encryption type for the zip item. (Inherited from ZipItem)
	LastAccessTime	Gets or sets the last access time for the zip item. (Inherited from ZipItem)
	LastAccessTimeUtc	Gets or sets the coordinated universal time value for the last access time for the zip item. (Inherited from ZipItem)
	LastWriteTime	Gets or sets the last write time for the zip item. (Inherited from ZipItem)
	LastWriteTimeUtc	Gets or sets the coordinated universal time value for the last write time for the zip item. (Inherited from ZipItem)
	Name	Gets or sets the name of the zip item. (Inherited from ZipItem)

	Password	Gets or sets a password used for encrypting the content of a zip item. (Inherited from ZipItem)
	UncompressedSize	Obtains the size of uncompressed data in the zip item. (Inherited from ZipItem)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Extract	Overloaded. Extracts the current archive item as a file into the specified directory and enables you to define the behavior in case of a file name conflict. (Inherited from ZipItem)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Open	Obtains the unzipped data stream of the zip item. (Inherited from ZipItem)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[ZipTextItem Members](#)[DevExpress.Compression Namespace](#)

Content Property

Gets or sets the content of the text zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property Content As [String](#)

C#

```
public string Content { get; set; }
```

Property Value

A string that is the content of the zip item in the archive.

Remarks

You can also use the [ZipArchive.AddText](#) method to create a text zip item, specify its content and add it to archive.

See Also

[ZipTextItem Class](#)

[ZipTextItem Members](#)

[DevExpress.Compression Namespace](#)

ContentEncoding Property

Gets or sets the character encoding of the text that is the content of a zip item.

Namespace: [DevExpress.Compression](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ContentEncoding As Encoding
```

C#

```
public Encoding ContentEncoding { get; set; }
```

Property Value

A [System.Text.Encoding](#) object that specifies the character encoding.

Remarks

Default character encoding for the [ZipTextItem](#) is **UTF8**. Use the **ContentEncoding** property to specify a different encoding.

See Also

[ZipTextItem Class](#)

[ZipTextItem Members](#)

[DevExpress.Compression Namespace](#)

DevExpress.Docs.Text

Contains classes used by text utilities for document processing.

Classes

	Class	Description
	EncodingDetector	Provides the capability to detect text encoding.
	NumberInWords	Provides access to static properties used to obtain a provider that converts numbers to words.

Interfaces

	Interface	Description
	INumberInWordsProvider	Converts numbers to cardinal and ordinal numerals in different languages.

Enumerations

	Enumeration	Description
	NumberCulture	Lists languages for which numbers can be converted to words.

EncodingDetector Class

Provides the capability to detect text encoding.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class EncodingDetector Inherits [Object](#)

C#

```
public class EncodingDetector : object
```

Remarks

Use the [Detect](#) method to obtain the [System.Text.Encoding](#) of the specified text stream or byte array.

Inheritance Hierarchy

[Object](#)

EncodingDetector

See Also

[EncodingDetector Members](#)


[DevExpress.Docs.Text Namespace](#)

EncodingDetector Members











Provides the capability to detect text encoding.

The following tables list the members exposed by the [EncodingDetector](#) type.

Public Constructors

	Name	Description
	EncodingDetector	Initializes a new instance of the EncodingDetector class with default settings.

Public Methods

	Name	Description
	AnalyseData	Performs encoding detection for the specified sequence of bytes in the byte array.
	BeginDetection	Initializes an internal list of detectors for individual encodings and sets the initial detector state.
	Detect	Overloaded. Detect character encoding of bytes in the specified range of an array.
	EndDetection	Returns the result of encoding detection.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[EncodingDetector Members](#)
[DevExpress.Docs.Text Namespace](#)

EncodingDetector Constructor

Initializes a new instance of the [EncodingDetector](#) class with default settings.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public EncodingDetector()
```

See Also

[EncodingDetector Class](#)

[EncodingDetector Members](#)











[DevExpress.Docs.Text Namespace](#)

EncodingDetector Methods

Provides the capability to detect text encoding.

The following tables list the members exposed by the [EncodingDetector](#) type.

Public Methods

	Name	Description
	AnalyseData	Performs encoding detection for the specified sequence of bytes in the byte array.
	BeginDetection	Initializes an internal list of detectors for individual encodings and sets the initial detector state.
	Detect	Overloaded. Detect character encoding of bytes in the specified range of an array.
	EndDetection	Returns the result of encoding detection.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[EncodingDetector Members](#)
[DevExpress.Docs.Text Namespace](#)

AnalyseData Method

Performs encoding detection for the specified sequence of bytes in the byte array.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AnalyseData(  
    ByVal buffer As Byte[],  
    ByVal from As Integer,  
    ByVal length As Integer  
) As Boolean
```

C#

```
public bool AnalyseData(  
    Byte[] buffer,  
    int from,  
    int length  
)
```

Parameters

buffer

An array of bytes representing text for which encoding is detected.

from

An integer specifying the array index that is the beginning of the range.

length

An integer specifying the number of bytes in the range.

Return Value

true if a certain encoding is recognized; otherwise, **false**.

Remarks

The **AnalyseData** method is called inside the [Detect](#) method.

See Also

[EncodingDetector Class](#)

[EncodingDetector Members](#)

[DevExpress.Docs.Text Namespace](#)

BeginDetection Method

Initializes an internal list of detectors for individual encodings and sets the initial detector state.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub BeginDetection()
```

C#

```
public void BeginDetection()
```

Remarks

The **BeginDetection** is called inside the [Detect](#) method.

See Also

[EncodingDetector Class](#)

[EncodingDetector Members](#)

[DevExpress.Docs.Text Namespace](#)

[Detect](#)

Detect Method

Detect character encoding of bytes in the specified range of an array.

Overload List

Name	Description
Encoding.Detect(Byte[] buffer)	Detect character encoding of bytes in the specified range of an array.
Encoding.Detect(Stream stream)	Detect character encoding of bytes in the specified stream.
Encoding.Detect(Stream stream, int maxByteCount)	Detect character encoding of bytes in the specified stream.
Encoding.Detect(Byte[] buffer, int from, int length)	Detect character encoding of bytes in the specified range of an array.
Encoding.Detect(Stream stream, int maxByteCount, bool keepPosition)	Detect character encoding of bytes in the specified stream.

See Also
[EncodingDetector Class](#)
[EncodingDetector Members](#)
[DevExpress.Docs.Text Namespace](#)
[EncodingDetector.Detect Overload List](#)

Detect character encoding of bytes in the specified range of an array.

Namespace: [DevExpress.Docs.Text](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Detect(  
    ByVal buffer As Byte[]  
) As Encoding
```

C#

```
public Encoding Detect(  
    Byte[] buffer  
)
```

Parameters

buffer
An array of bytes representing text for which encoding is detected.

Return Value

An [System.Text.Encoding](#) object that is the detected encoding.

See Also
[EncodingDetector Class](#)
[EncodingDetector Members](#)
[DevExpress.Docs.Text Namespace](#)
[EncodingDetector.Detect Overload List](#)

Detect character encoding of bytes in the specified stream.

Namespace: [DevExpress.Docs.Text](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Detect(  
    ByVal stream As Stream  
) As Encoding
```

C#

```
public Encoding Detect(  
    Stream stream  
)
```

Parameters

stream

A [System.IO.Stream](#) containing characters for which encoding is detected.

Return Value

An [System.Text.Encoding](#) object that is the detected encoding.

Remarks

;

See Also

[EncodingDetector Class](#)

[EncodingDetector Members](#)

[DevExpress.Docs.Text Namespace](#)

[EncodingDetector.Detect Overload List](#)

Detect character encoding of bytes in the specified stream.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function Detect(  
    ByVal stream As Stream,  
    ByVal maxByteCount As Integer  
) As Encoding
```

C#

```
public Encoding Detect(  
    Stream stream,  
    int maxByteCount  
)
```

Parameters

stream

A [System.IO.Stream](#) containing characters for which encoding is detected.

maxByteCount

An integer specifying the maximum number of bytes used for encoding detection.

Return Value

An [System.Text.Encoding](#) object that is the detected encoding.

Remarks

;

See Also

[EncodingDetector Class](#)

[EncodingDetector Members](#)

[DevExpress.Docs.Text Namespace](#)

[EncodingDetector.Detect Overload List](#)

Detect character encoding of bytes in the specified range of an array.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function Detect(
    ByVal buffer As Byte[],
    ByVal from As Integer,
    ByVal length As Integer
) As Encoding
```

C#

```
public Encoding Detect(
    Byte[] buffer,
    int from,
    int length
)
```

Parameters*buffer*

An array of bytes representing text for which encoding is detected.

from

An integer specifying the array index that is the beginning of the range.

length

An integer specifying the number of bytes in the range.

Return Value

An [System.Text.Encoding](#) object that is the detected encoding.

See Also

[EncodingDetector Class](#)

[EncodingDetector Members](#)

[DevExpress.Docs.Text Namespace](#)

[EncodingDetector.Detect Overload List](#)

Detect character encoding of bytes in the specified stream.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function Detect(
    ByVal stream As Stream,
    ByVal maxByteCount As Integer,
    ByVal keepPosition As Boolean
) As Encoding
```

C#

```
public Encoding Detect(
    Stream stream,
    int maxByteCount,
    bool keepPosition
)
```

Parameters*stream*

A [System.IO.Stream](#) containing characters for which encoding is detected.

maxByteCount

An integer specifying the maximum number of bytes used for encoding detection.

keepPosition

true to start detection from the **System.IO.Stream.Position** offset; otherwise, **false**.

Return Value

An [System.Text.Encoding](#) object that is the detected encoding.

See Also

[EncodingDetector Class](#)

[EncodingDetector Members](#)

[DevExpress.Docs.Text Namespace](#)
[EncodingDetector.Detect Overload List](#)

EndDetection Method

Returns the result of encoding detection.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function EndDetection() As Encoding
```

C#

```
public Encoding EndDetection()
```

Return Value

An [System.Text.Encoding](#) object that is the detected encoding.

Remarks

The **EndDetection** is called inside the [Detect](#) method.

See Also

[EncodingDetector Class](#)

[EncodingDetector Members](#)

[DevExpress.Docs.Text Namespace](#)

[Detect](#)

INumberInWordsProvider Interface

Converts numbers to cardinal and ordinal numerals in different languages.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Interface INumberInWordsProvider
```

C#

```
public interface INumberInWordsProvider
```

Remarks

Use the [NumberInWords.Cardinal](#) and [NumberInWords.Ordinal](#) static methods to obtain the converter that can be used to convert numbers to cardinal and ordinal numerals respectively.

Inheritance Hierarchy

See Also

[INumberInWordsProvider Members](#)

[DevExpress.Docs.Text Namespace](#)

INumberInWordsProvider Members

Converts numbers to cardinal and ordinal numerals in different languages.

The following tables list the members exposed by the [INumberInWordsProvider](#) type.

Public Methods

	Name	Description
	ConvertToText	Overloaded. Converts a number to text.

See Also

[INumberInWordsProvider Members](#)


[DevExpress.Docs.Text Namespace](#)

INumberInWordsProvider Methods

Converts numbers to cardinal and ordinal numerals in different languages.

The following tables list the members exposed by the [INumberInWordsProvider](#) type.

Public Methods

	Name	Description
	ConvertToText	Overloaded. Converts a number to text.

See Also

[INumberInWordsProvider Members](#)

[DevExpress.Docs.Text Namespace](#)

ConvertToText Method

Converts a number to text.

Overload List

Name	Description
string ConvertToText(Int64 value)	Converts a number to text.
string ConvertToText(Int64 value, CultureInfo culture)	Converts a number to text.
string ConvertToText(Int64 value, NumberCulture language)	Converts a number to text.

See Also
[INumberInWordsProvider Interface](#)
[INumberInWordsProvider Members](#)
[DevExpress.Docs.Text Namespace](#)
[INumberInWordsProvider.ConvertToText Overload List](#)

Converts a number to text.

Namespace: [DevExpress.Docs.Text](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
Function ConvertToText(
 ByVal value As [Int64](#)
) As [String](#)
C#
[string](#) ConvertToText(
 [Int64](#) value
)

Parameters
value
A [System.Int64](#) value to be converted to text.

Return Value
A string that is the numeral corresponding to the specified number.

Remarks
Use the [NumberInWords.Ordinal](#) or [NumberInWords.Cardinal](#) static methods to obtain the converter for cardinal and ordinal numerals respectively.

See Also
[INumberInWordsProvider Interface](#)
[INumberInWordsProvider Members](#)
[DevExpress.Docs.Text Namespace](#)
[INumberInWordsProvider.ConvertToText Overload List](#)

Converts a number to text.

Namespace: [DevExpress.Docs.Text](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
Function ConvertToText(
 ByVal value As [Int64](#),
 ByVal culture As [CultureInfo](#)
) As [String](#)

C#

```
string ConvertToText(  
    Int64 value,  
    CultureInfo culture  
)
```

Parameters

value

A [System.Int64](#) value to be converted to text.

culture

A [System.Globalization.CultureInfo](#) object that specifies the language of numerals.

Return Value

A string that is the numeral corresponding to the specified number.

Remarks

Use the [NumberInWords.Ordinal](#) or [NumberInWords.Cardinal](#) static methods to obtain the converter for cardinal and ordinal numerals respectively.

If the language of the specified culture is not listed in the [NumberCulture](#) enumeration, English numerals are returned.

See Also

[INumberInWordsProvider Interface](#)

[INumberInWordsProvider Members](#)

[DevExpress.Docs.Text Namespace](#)

[INumberInWordsProvider.ConvertToText Overload List](#)

Converts a number to text.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Function ConvertToText(  
    ByVal value As Int64,  
    ByVal language As NumberCulture  
) As String
```

C#

```
string ConvertToText(  
    Int64 value,  
    NumberCulture language  
)
```

Parameters

value

A [System.Int64](#) value to be converted to text.

language

A [NumberCulture](#) enumeration member that specifies the language of the numerals.

Return Value

A string that is the numeral corresponding to the specified number.

Remarks

Use the [NumberInWords.Ordinal](#) or [NumberInWords.Cardinal](#) static methods to obtain the converter for cardinal and ordinal numerals respectively.

See Also

[INumberInWordsProvider Interface](#)

[INumberInWordsProvider Members](#)

[DevExpress.Docs.Text Namespace](#)

[INumberInWordsProvider.ConvertToText Overload List](#)

NumberCulture Enumeration

Lists languages for which numbers can be converted to words.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Enum NumberCulture

C#

public enum NumberCulture

Members

Name	Description
English	English numerals.
English_UnitedKingdom	English numerals (hundreds and tens are separated by "and").
French	French numerals.
German	German numerals.
Greek	Greek numerals.
Hindi	Hindi numerals.
Italian	Italian numerals.
LatinLetter	Latin Letter numerals.
Portuguese	Portuguese numerals.
Roman	Roman numerals.
Russian	Russian numerals.
RussianLetter	Russian Letter numerals.
Spanish	Spanish numerals.
Swedish	Swedish numerals.
Thai	Thai numerals.
Turkish	Turkish numerals.
Ukrainian	Ukrainian numerals.

See Also

[DevExpress.Docs.Text Namespace](#)

NumberInWords Class

Provides access to static properties used to obtain a provider that converts numbers to words.

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

`NotInheritable Public Class NumberInWords Inherits Object`

C#

`public sealed abstract class NumberInWords : object`

Remarks

Use the [Ordinal](#) property to obtain a word representing a number in sequential order, and the [Cardinal](#) property to obtain a word representing a natural number.



The following code snippet obtains two strings - "Five hundred sixty-eighth" and "F

NumberInWords Members







Provides access to static properties used to obtain a provider that converts numbers to words.

The following tables list the members exposed by the [NumberInWords](#) type.

Public Properties

	Name	Description
	Cardinal	Obtains the numbers-to-words converter for cardinal numbers (numbers which refer to the size of a group, simple numerals).
	Ordinal	Obtains the numbers-to-words converter for ordinal numbers (numbers which refer to a position in a series).

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also



[NumberInWords Members](#)
[DevExpress.Docs.Text Namespace](#)

NumberInWords Properties







Provides access to static properties used to obtain a provider that converts numbers to words.

The following tables list the members exposed by the [NumberInWords](#) type.

Public Properties

	Name	Description
	Cardinal	Obtains the numbers-to-words converter for cardinal numbers (numbers which refer to the size of a group, simple numerals).
	Ordinal	Obtains the numbers-to-words converter for ordinal numbers (numbers which refer to a position in a series).

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[NumberInWords Members](#)
[DevExpress.Docs.Text Namespace](#)

Cardinal Property

Obtains the numbers-to-words converter for cardinal numbers (numbers which refer to the size of a group, simple numerals).

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Cardinal As INumberInWordsProvider
```

C#

```
public static INumberInWordsProvider Cardinal { get; }
```

Property Value

An [INumberInWordsProvider](#) object that is a numbers-to-words converter.

Remarks

The **Cardinal** static method creates an object that can be used to convert numbers to simple numerals which represent cardinal numbers. To accomplish this, call the [INumberInWordsProvider.ConvertToText](#) method.

See Also

[NumberInWords Class](#)

[NumberInWords Members](#)

[DevExpress.Docs.Text Namespace](#)

Ordinal Property

Obtains the numbers-to-words converter for ordinal numbers (numbers which refer to a position in a series).

Namespace: [DevExpress.Docs.Text](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Ordinal As INumberInWordsProvider
```

C#

```
public static INumberInWordsProvider Ordinal { get; }
```

Property Value

An [INumberInWordsProvider](#) object that is a numbers-to-words converter.

Remarks

The **Ordinal** static method creates an object that can be used to convert numbers to numerals which represent ordinal (first, second, third, etc.) numbers. To accomplish this, call the [INumberInWordsProvider.ConvertToText](#) method.

See Also

[NumberInWords Class](#)

[NumberInWords Members](#)

[DevExpress.Docs.Text Namespace](#)

DevExpress.Pdf

Contains classes and enumerations that are used to implement the main functionality of WinForms and WPF PDF Viewers, and the PDF Document API.

Classes

	Class	Description
	PdfDocumentProcessor	A non-visual component that allows you to easily generate PDF files from scratch and manipulate existing PDF documents. See PDF Document API .
	PdfPageWord	An individual word corresponding to a specific PDF page.
	PdfViewerExtensions	Defines extension methods that are used to extend the functionality of the WinForms PDF Viewer and WPF PDF Viewer.

PdfDocumentProcessor Class

A non-visual component that allows you to easily generate PDF files from scratch and manipulate existing PDF documents. See [PDF Document API](#).

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class PdfDocumentProcessor Inherits PdfDisposableObject
```

C#

```
public class PdfDocumentProcessor : PdfDisposableObject
```

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

To generate a document using the Document Creation API:

- Create an empty document with no pages by calling one of the [CreateEmptyDocument](#) overload methods (e.g., using a file path).
- Create PDF graphics represented by an instance of the `DevExpress.Pdf.PdfGraphics` class, calling the [CreateGraphics](#) method. To access `DevExpress.Pdf.PdfGraphics`, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer.")
            }
        }
    }
}
```

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace
```

Inheritance Hierarchy

[Object](#)

DevExpress.Pdf.Native.PdfDisposableObject

PdfDocumentProcessor

See Also

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PDF Document API](#)

PdfDocumentProcessor Members








A non-visual component that allows you to easily generate PDF files from scratch and manipulate existing PDF documents. See [PDF Document API](#).

The following tables list the members exposed by the [PdfDocumentProcessor](#) type.







Public Constructors

	Name	Description
	PdfDocumentProcessor	Initializes a new instance of the PdfDocumentProcessor class with default settings.

Public Properties

	Name	Description
	Document	Returns the current document.
	ImageCacheSize	Specifies the size of the image cache (in megabytes).
	MaxPrintingDpi	Specifies the maximum DPI (dots per inch) value allowed for document printing.
	PasswordAttemptsLimit	Specifies the maximum number of allowed attempts to enter the PDF file's security password.
	RenderPageContentWithDirectX	Gets or sets whether to render page content with DirectX.
	ShowImagePlaceholder	Obsolete. Gets or sets a value which indicates whether to show placeholders for images in unsupported image formats.
	Text	Returns the PDF text.

Public Methods

	Name	Description
	AddFormFields	Overloaded. Adds interactive form fields to a PDF document.
	AddNewPage	Adds a new page with the specified page size to the current document.
	AddTextMarkupAnnotation	Overloaded. Adds a text markup annotation to the specified area on the page.
	AppendDocument	Overloaded. Appends a PDF document located at the specified file stream to the end of the current document starting from the new page.
	ApplyFormData	Fills an interactive form with data.
	AttachFile	Attaches a file to the PDF document.

	CheckFormFieldNameCollisions	Overloaded. Checks interactive form fields to find a collision in the form field names.
	CloseDocument	Closes the current document.
	CreateBitmap	Exports a PDF page to a bitmap image.
	CreateDestination	Overloaded. Creates a destination for targets in the document (e.g., bookmarks) using a page number, and zoom factor.
	CreateEmptyDocument	Overloaded. Creates an empty PDF document with no pages.
	CreateGraphics	Creates a new instance of the <code>DevExpress.Pdf.PdfGraphics</code> used as a drawing context that can draw graphics on a PDF document.
	CreateTiff	Overloaded. Exports a PDF document to a TIFF image using a stream, the image's largest edge length, and page numbers.
	DeleteAttachment	Returns a value indicating if the attachment is deleted from a PDF document.
	DeleteMarkupAnnotation	Deletes a markup annotation from a page.
	DeleteMarkupAnnotations	Deletes markup annotations specified in the <code>DevExpress.Pdf.PdfMarkupAnnotationData</code> collection.
	DeletePage	Deletes the specified page from the current document.
	Dispose	(Inherited from <code>DevExpress.Pdf.Native.PdfDisposableObject</code>)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Export	Overloaded. Exports interactive form data to the file using the specified form data format.
	FindText	Overloaded. Searches for the specified text in the current document with default parameters.
	FlattenForm	Flattens an entire interactive form.
	FlattenFormField	Flattens a specific form field on an interactive form by its name.

	GetFormData	Returns an object containing values of interactive form data fields.
	GetFormFieldNames	Returns a list of interactive field names in a document.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetImages	Overloaded. Retrieves the images found within the specified document positions using image resolution.
	GetMarkupAnnotationData	Retrieves all text markup annotations from a page in a PDF document.
	GetPageText	Obtains text from the specified page.
	GetText	Overloaded. Retrieves the text found within the specified document positions.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Import	Overloaded. Imports interactive form data from the specified stream.
	InsertNewPage	Inserts a new page with a specified page number and page size into the document.
	LoadDocument	Overloaded. Opens a PDF document from the specified stream.
	NextWord	Returns the next word in a PDF document.
	PrevWord	Returns the previous word in a PDF document.
	Print	Overloaded. Obsolete. Prints the current document, using the specified settings.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveForm	Removes all interactive form fields from a PDF document.
	RemoveFormField	Removes an interactive form field from a document using the field name.
	RenderNewPage	Overloaded. Adds a new page with specified page size and created graphics to a document.
	ResetFormData	Resets all fields of the interactive form to their default values.
	SaveDocument	Overloaded. Saves the current document to the specified file path.

	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
--	--------------------------	--

Public Events

	Name	Description
	PasswordRequested	Occurs when requesting a security password to open a protected PDF file.
	PrintPage	Occurs when the document page is printed.
	QueryPageSettings	Occurs immediately before the PrintPage event.

See Also
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PDF Document API](#)

PdfDocumentProcessor Constructor

Initializes a new instance of the [PdfDocumentProcessor](#) class with default settings.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public PdfDocumentProcessor()
```

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the `DevExpress.Pdf.PdfFormData.Value` property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#

```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic

```
(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using
```


See Also[PdfDocumentProcessor Class](#)[PdfDocumentProcessor Members](#)[DevExpress.Pdf Namespace](#)

PdfDocumentProcessor Properties








A non-visual component that allows you to easily generate PDF files from scratch and manipulate existing PDF documents. See [PDF Document API](#).

The following tables list the members exposed by the [PdfDocumentProcessor](#) type.







Public Constructors

	Name	Description
	PdfDocumentProcessor	Initializes a new instance of the PdfDocumentProcessor class with default settings.

Public Properties

	Name	Description
	Document	Returns the current document.
	ImageCacheSize	Specifies the size of the image cache (in megabytes).
	MaxPrintingDpi	Specifies the maximum DPI (dots per inch) value allowed for document printing.
	PasswordAttemptsLimit	Specifies the maximum number of allowed attempts to enter the PDF file's security password.
	RenderPageContentWithDirectX	Gets or sets whether to render page content with DirectX.
	ShowImagePlaceholder	Obsolete. Gets or sets a value which indicates whether to show placeholders for images in unsupported image formats.
	Text	Returns the PDF text.

Public Methods

	Name	Description
	AddFormFields	Overloaded. Adds interactive form fields to a PDF document.
	AddNewPage	Adds a new page with the specified page size to the current document.
	AddTextMarkupAnnotation	Overloaded. Adds a text markup annotation to the specified area on the page.
	AppendDocument	Overloaded. Appends a PDF document located at the specified file stream to the end of the current document starting from the new page.
	ApplyFormData	Fills an interactive form with data.
	AttachFile	Attaches a file to the PDF document.

	CheckFormFieldNameCollisions	Overloaded. Checks interactive form fields to find a collision in the form field names.
	CloseDocument	Closes the current document.
	CreateBitmap	Exports a PDF page to a bitmap image.
	CreateDestination	Overloaded. Creates a destination for targets in the document (e.g., bookmarks) using a page number, and zoom factor.
	CreateEmptyDocument	Overloaded. Creates an empty PDF document with no pages.
	CreateGraphics	Creates a new instance of the <code>DevExpress.Pdf.PdfGraphics</code> used as a drawing context that can draw graphics on a PDF document.
	CreateTiff	Overloaded. Exports a PDF document to a TIFF image using a stream, the image's largest edge length, and page numbers.
	DeleteAttachment	Returns a value indicating if the attachment is deleted from a PDF document.
	DeleteMarkupAnnotation	Deletes a markup annotation from a page.
	DeleteMarkupAnnotations	Deletes markup annotations specified in the <code>DevExpress.Pdf.PdfMarkupAnnotationData</code> collection.
	DeletePage	Deletes the specified page from the current document.
	Dispose	(Inherited from <code>DevExpress.Pdf.Native.PdfDisposableObject</code>)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Export	Overloaded. Exports interactive form data to the file using the specified form data format.
	FindText	Overloaded. Searches for the specified text in the current document with default parameters.
	FlattenForm	Flattens an entire interactive form.
	FlattenFormField	Flattens a specific form field on an interactive form by its name.

	GetFormData	Returns an object containing values of interactive form data fields.
	GetFormFieldNames	Returns a list of interactive field names in a document.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetImages	Overloaded. Retrieves the images found within the specified document positions using image resolution.
	GetMarkupAnnotationData	Retrieves all text markup annotations from a page in a PDF document.
	GetPageText	Obtains text from the specified page.
	GetText	Overloaded. Retrieves the text found within the specified document positions.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Import	Overloaded. Imports interactive form data from the specified stream.
	InsertNewPage	Inserts a new page with a specified page number and page size into the document.
	LoadDocument	Overloaded. Opens a PDF document from the specified stream.
	NextWord	Returns the next word in a PDF document.
	PrevWord	Returns the previous word in a PDF document.
	Print	Overloaded. Obsolete. Prints the current document, using the specified settings.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveForm	Removes all interactive form fields from a PDF document.
	RemoveFormField	Removes an interactive form field from a document using the field name.
	RenderNewPage	Overloaded. Adds a new page with specified page size and created graphics to a document.
	ResetFormData	Resets all fields of the interactive form to their default values.
	SaveDocument	Overloaded. Saves the current document to the specified file path.

	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
--	--------------------------	--

Public Events

	Name	Description
	PasswordRequested	Occurs when requesting a security password to open a protected PDF file.
	PrintPage	Occurs when the document page is printed.
	QueryPageSettings	Occurs immediately before the PrintPage event.

See Also
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PDF Document API](#)

Document Property

Returns the current document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Document As PdfDocument
```

C#

```
public PdfDocument Document { get; }
```

Property Value

A DevExpress.Pdf.PdfDocument object.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

ImageCacheSize Property

Specifies the size of the image cache (in megabytes).

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ImageCacheSize As Int64
```

C#

```
public Int64 ImageCacheSize { get; set; }
```

Property Value

An integer value, specifying the image cache size (in megabytes).

Remarks

If this property is set to **0**, the image cache size is unlimited.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

MaxPrintingDpi Property

Specifies the maximum DPI (dots per inch) value allowed for document printing.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property MaxPrintingDpi As Integer
```

C#

```
public int MaxPrintingDpi { get; set; }
```

Property Value

An integer value. The zero value indicates that the printing DPI is not limited.

Remarks

If a printer unexpectedly produces empty pages when printing a PDF document, you can avoid this by setting the **MaxPrintingDpi** property to **300**.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

PasswordAttemptsLimit Property

Specifies the maximum number of allowed attempts to enter the PDF file's security password.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property PasswordAttemptsLimit As Integer
```

C#

```
public int PasswordAttemptsLimit { get; set; }
```

Property Value

An integer value.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

RenderPageContentWithDirectX Property

Gets or sets whether to render page content with DirectX.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property RenderPageContentWithDirectX As Boolean
```

C#

```
public bool RenderPageContentWithDirectX { get; set; }
```

Property Value

true, to enable DirectX page content rendering; **false**, to render page content with GDI/GDI+. The default is **false**.

Remarks

Note that DirectX applications require Platform Update for Windows 7, Windows 8 or newer installed on client machines.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

ShowImagePlaceholder Property

NOTE: This member is now obsolete.

This API is now obsolete. You don't need the ShowImagePlaceholder property anymore.

Gets or sets a value which indicates whether to show placeholders for images in unsupported image formats.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property ShowImagePlaceholder As Boolean
```

C#

```
public bool ShowImagePlaceholder { get; set; }
```

Property Value

true to show image placeholders in unsupported image formats; otherwise **false**.

Remarks

The **ShowImagePlaceholder** property can be used with the [CreateTiff](#), [CreateBitmap](#) or [Print](#) method.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

Text Property

Returns the PDF text.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Text As String
```

C#

```
public string Text { get; }
```

Property Value

A [System.String](#) value.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

PdfDocumentProcessor Events

A non-visual component that allows you to easily generate PDF files from scratch and manipulate existing PDF documents. See [PDF Document API](#).

The following tables list the members exposed by the [PdfDocumentProcessor](#) type.

Public Events

	Name	Description
	PasswordRequested	Occurs when requesting a security password to open a protected PDF file.
	PrintPage	Occurs when the document page is printed.
	QueryPageSettings	Occurs immediately before the PrintPage event.

See Also
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PDF Document API](#)

PasswordRequested Event

Occurs when requesting a security password to open a protected PDF file.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public PasswordRequested As PdfPasswordRequestedEventHandler
```

C#

```
public event PdfPasswordRequestedEventHandler PasswordRequested
```

Event Data

The event handler receives an argument of type PdfPasswordRequestedEventArgs containing data related to this event.

Remarks

To interrupt the password request process, the DevExpress.Pdf.PdfPasswordRequestedEventArgs.PasswordString property value should be set to **null**.

An invalid password exception is raised when calling the [LoadDocument](#) method if the DevExpress.Pdf.PdfPasswordRequestedEventArgs.PasswordString property has either the **null** value or when the password request limit is expired (all requests return an incorrect password).

To change the maximum number of allowed attempts to enter the PDF file's security password, use the [PasswordAttemptsLimit](#) property (the default value is **1**).

Example

The following code illustrates the use of the [PasswordRequested](#) event.

C#

```
using DevExpress.Pdf;
namespace RequestPassword {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) {
                // Handle the PasswordRequested event.
                pdfDocumentProcessor.PasswordRequested += PdfDocumentProcessor_PasswordRequested;
                // Load a protected PDF document.
                pdfDocumentProcessor.LoadDocument(@"..\..\ProtectedDocument.pdf");
                // Unsubscribe from PasswordRequested event.
                pdfDocumentProcessor.PasswordRequested -= PdfDocumentProcessor_PasswordRequested;
                // Save the document.
                pdfDocumentProcessor.SaveDocument(@"..\..\Result.pdf");
            }
        }
        private static void PdfDocumentProcessor_PasswordRequested(object sender, PdfPasswordRequestedEventArgs e) {
            e.PasswordString = "password";
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
Namespace RequestPassword
    Class Program
        Private Shared Sub Main(args As String())
            Using pdfDocumentProcessor As New PdfDocumentProcessor()
                ' Handle the PasswordRequested event.
                AddHandler pdfDocumentProcessor.PasswordRequested, AddressOf PdfDocumentProcessor_PasswordRequested
                ' Load a protected PDF document.
                pdfDocumentProcessor.LoadDocument("..\\..\\ProtectedDocument.pdf")
                ' Unsubscribe from PasswordRequested event.
                RemoveHandler pdfDocumentProcessor.PasswordRequested, AddressOf PdfDocumentProcessor_PasswordRequested
                ' Save the document.
                pdfDocumentProcessor.SaveDocument("..\\..\\Result.pdf")
            End Using
        End Sub
        Private Shared Sub PdfDocumentProcessor_PasswordRequested(sender As Object, e As PdfPasswordRequestEventArgs)
            e.PasswordString = "password"
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)

PrintPage Event

Occurs when the document page is printed.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public PrintPage As PdfPrintPageEventHandler
```

C#

```
public event PdfPrintPageEventHandler PrintPage
```

Event Data

The event handler receives an argument of type PdfPrintPageEventArgs containing data related to this event.

Remarks

Use this event to perform actions when the document is sent to a printer. The page number of the currently printed page and the total number of pages which were sent to the printer can be accessed via the DevExpress.Pdf.PdfPrintPageEventArgs.PageNumber and DevExpress.Pdf.PdfPrintPageEventArgs.PageCount properties, respectively.

The print job can also be canceled by setting the **Cancel** property to **true** for the **PrintPageEventArgs**. For more information about other properties of **PrintPageEventArgs**, see the [PrintPageEventArgs Class](#) MSDN topic.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T334688>.

This example shows how to customize print output and change settings for a page to be printed.

A document is printed with the specified initial printer settings using the [Print](#) method. In this example, the page numbers that should be printed in a document are specified using the DevExpress.Pdf.PdfPrinterSettings.PageNumbers property.

When a page is printed, page settings for this page are queried via the [QueryPageSettings](#) event.

In this example, the second page of a document is printed in landscape orientation by setting the **PdfQueryPageSettingsEventArgs.PageSettings.Landscape** property to **true** when the [QueryPageSettings](#) event is handled.

When the document is sent to a printer, the [PrintPage](#) event is raised.

To draw an additional image on a printed page, the **Graphics** property of the DevExpress.Pdf.PdfPrintPageEventArgs is used when the [PrintPage](#) event is handled. This image is drawn on each page by calling the **Graphics.DrawImage** method.

C#

```

(Program.cs)
using DevExpress.Pdf;
using System.Drawing;
namespace CustomizePrintSettings {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor instance and load a PDF into it.
            using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
                documentProcessor.LoadDocument(@"..\..\Demo.pdf");
                // Declare the PDF printer settings.
                PdfPrinterSettings settings = new PdfPrinterSettings();
                // Specify the page numbers to be printed.
                settings.PageNumbers = new int[] { 1, 2, 3, 4, 5, 6 };
                // Handle the PrintPage event to specify print output.
                documentProcessor.PrintPage += OnPrintPage;
                // Handle the QueryPageSettings event to customize settings for a page to be printed.
                documentProcessor.QueryPageSettings += OnQueryPageSettings;
                // Print the document using the specified printer settings.
                documentProcessor.Print(settings);
                // Unsubscribe from PrintPage and QueryPageSettings events.
                documentProcessor.PrintPage -= OnPrintPage;
                documentProcessor.QueryPageSettings -= OnQueryPageSettings;
            }
        }
        private static void OnQueryPageSettings(object sender, PdfQueryPageSettingsEventArgs e) {
            // Print the second page in landscape size.
            if (e.PageNumber == 2) {
                e.PageSettings.Landscape = true;
            }
            else e.PageSettings.Landscape = false;
        }
        // Specify what happens when the PrintPage event is raised.
        private static void OnPrintPage(object sender, PdfPrintPageEventArgs e) {
            // Draw a picture on each printed page.
            using (Bitmap image = new Bitmap(@"..\..\DevExpress.png"))
                e.Graphics.DrawImage(image, new RectangleF(10, 30, image.Width / 2, image.Height / 2));
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System.Drawing
Namespace CustomizePrintSettings
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor instance and load a PDF into it.
            Using documentProcessor As New PdfDocumentProcessor()
                documentProcessor.LoadDocument("../..\..\Demo.pdf")
                ' Declare the PDF printer settings.
                Dim settings As New PdfPrinterSettings()
                ' Specify the page numbers to be printed.
                settings.PageNumbers = New Integer() { 1, 2, 3, 4, 5, 6 }
                ' Handle the PrintPage event to specify print output.
                AddHandler documentProcessor.PrintPage, AddressOf OnPrintPage
                ' Handle the QueryPageSettings event to customize settings for a page to be printed.
                AddHandler documentProcessor.QueryPageSettings, AddressOf OnQueryPageSettings
                ' Print the document using the specified printer settings.
                documentProcessor.Print(settings)
                ' Unsubscribe from PrintPage and QueryPageSettings events.
                RemoveHandler documentProcessor.PrintPage, AddressOf OnPrintPage
                RemoveHandler documentProcessor.QueryPageSettings, AddressOf OnQueryPageSettings
            End Using
        End Sub
        Private Shared Sub OnQueryPageSettings(ByVal sender As Object, ByVal e As PdfQueryPageSettingsEventArgs)
            ' Print the second page in landscape size.
            If e.PageNumber = 2 Then
                e.PageSettings.Landscape = True
            Else
                e.PageSettings.Landscape = False
            End If
        End Sub
        ' Specify what happens when the PrintPage event is raised.
        Private Shared Sub OnPrintPage(ByVal sender As Object, ByVal e As PdfPrintPageEventArgs)
            ' Draw a picture on each printed page.
            Using image As New Bitmap("../..\..\DevExpress.png")
                e.Graphics.DrawImage(image, New RectangleF(10, 30, image.Width \ 2, image.Height \ 2))
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

QueryPageSettings Event

Occurs immediately before the [PrintPage](#) event.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public QueryPageSettings As PdfQueryPageSettingsEventHandler
```

C#

```
public event PdfQueryPageSettingsEventHandler QueryPageSettings
```

Event Data

The event handler receives an argument of type PdfQueryPageSettingsEventArgs containing data related to this event.

Remarks

You can print each page of a document using different page settings. To access the page settings, use the corresponding property of the **QueryPageSettingsEventArgs**. For more information about other properties of **QueryPageSettingsEventArgs**, see the [QueryPageSettingsEventArgs Class](#) MSDN topic.

You can also obtain the page number and page size for each page in a document using the DevExpress.Pdf.PdfQueryPageSettingsEventArgs.PageNumber and DevExpress.Pdf.PdfQueryPageSettingsEventArgs.PageSize properties, correspondingly.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T334688>.

This example shows how to customize print output and change settings for a page to be printed.

A document is printed with the specified initial printer settings using the [Print](#) method. In this example, the page numbers that should be printed in a document are specified using the DevExpress.Pdf.PdfPrinterSettings.PageNumbers property.

When a page is printed, page settings for this page are queried via the [QueryPageSettings](#) event.

In this example, the second page of a document is printed in landscape orientation by setting the **PdfQueryPageSettingsEventArgs.PageSettings.Landscape** property to **true** when the [QueryPageSettings](#) event is handled.

When the document is sent to a printer, the [PrintPage](#) event is raised.

To draw an additional image on a printed page, the **Graphics** property of the DevExpress.Pdf.PdfPrintPageEventArgs is used when the [PrintPage](#) event is handled. This image is drawn on each page by calling the **Graphics.DrawImage** method.

C#

```

(Program.cs)
using DevExpress.Pdf;
using System.Drawing;
namespace CustomizePrintSettings {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor instance and load a PDF into it.
            using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
                documentProcessor.LoadDocument(@"..\..\Demo.pdf");
                // Declare the PDF printer settings.
                PdfPrinterSettings settings = new PdfPrinterSettings();
                // Specify the page numbers to be printed.
                settings.PageNumbers = new int[] { 1, 2, 3, 4, 5, 6 };
                // Handle the PrintPage event to specify print output.
                documentProcessor.PrintPage += OnPrintPage;
                // Handle the QueryPageSettings event to customize settings for a page to be printed.
                documentProcessor.QueryPageSettings += OnQueryPageSettings;
                // Print the document using the specified printer settings.
                documentProcessor.Print(settings);
                // Unsubscribe from PrintPage and QueryPageSettings events.
                documentProcessor.PrintPage -= OnPrintPage;
                documentProcessor.QueryPageSettings -= OnQueryPageSettings;
            }
        }
        private static void OnQueryPageSettings(object sender, PdfQueryPageSettingsEventArgs e) {
            // Print the second page in landscape size.
            if (e.PageNumber == 2) {
                e.PageSettings.Landscape = true;
            }
            else e.PageSettings.Landscape = false;
        }
        // Specify what happens when the PrintPage event is raised.
        private static void OnPrintPage(object sender, PdfPrintPageEventArgs e) {
            // Draw a picture on each printed page.
            using (Bitmap image = new Bitmap(@"..\..\DevExpress.png"))
                e.Graphics.DrawImage(image, new RectangleF(10, 30, image.Width / 2, image.Height / 2));
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System.Drawing
Namespace CustomizePrintSettings
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor instance and load a PDF into it.
            Using documentProcessor As New PdfDocumentProcessor()
                documentProcessor.LoadDocument("../..\..\Demo.pdf")
                ' Declare the PDF printer settings.
                Dim settings As New PdfPrinterSettings()
                ' Specify the page numbers to be printed.
                settings.PageNumbers = New Integer() { 1, 2, 3, 4, 5, 6 }
                ' Handle the PrintPage event to specify print output.
                AddHandler documentProcessor.PrintPage, AddressOf OnPrintPage
                ' Handle the QueryPageSettings event to customize settings for a page to be printed.
                AddHandler documentProcessor.QueryPageSettings, AddressOf OnQueryPageSettings
                ' Print the document using the specified printer settings.
                documentProcessor.Print(settings)
                ' Unsubscribe from PrintPage and QueryPageSettings events.
                RemoveHandler documentProcessor.PrintPage, AddressOf OnPrintPage
                RemoveHandler documentProcessor.QueryPageSettings, AddressOf OnQueryPageSettings
            End Using
        End Sub
        Private Shared Sub OnQueryPageSettings(ByVal sender As Object, ByVal e As PdfQueryPageSettingsEventArgs)
            ' Print the second page in landscape size.
            If e.PageNumber = 2 Then
                e.PageSettings.Landscape = True
            Else
                e.PageSettings.Landscape = False
            End If
        End Sub
        ' Specify what happens when the PrintPage event is raised.
        Private Shared Sub OnPrintPage(ByVal sender As Object, ByVal e As PdfPrintPageEventArgs)
            ' Draw a picture on each printed page.
            Using image As New Bitmap("../..\..\DevExpress.png")
                e.Graphics.DrawImage(image, New RectangleF(10, 30, image.Width \ 2, image.Height \ 2))
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

PdfDocumentProcessor Methods

A non-visual component that allows you to easily generate PDF files from scratch and manipulate existing PDF documents. See [PDF Document API](#).

The following tables list the members exposed by the [PdfDocumentProcessor](#) type.

Public Methods

	Name	Description
	AddFormFields	Overloaded. Adds interactive form fields to a PDF document.
	AddNewPage	Adds a new page with the specified page size to the current document.
	AddTextMarkupAnnotation	Overloaded. Adds a text markup annotation to the specified area on the page.
	AppendDocument	Overloaded. Appends a PDF document located at the specified file stream to the end of the current document starting from the new page.
	ApplyFormData	Fills an interactive form with data.
	AttachFile	Attaches a file to the PDF document.
	CheckFormFieldNameCollisions	Overloaded. Checks interactive form fields to find a collision in the form field names.
	CloseDocument	Closes the current document.
	CreateBitmap	Exports a PDF page to a bitmap image.
	CreateDestination	Overloaded. Creates a destination for targets in the document (e.g., bookmarks) using a page number, and zoom factor.
	CreateEmptyDocument	Overloaded. Creates an empty PDF document with no pages.
	CreateGraphics	Creates a new instance of the DevExpress.Pdf.PdfGraphics used as a drawing context that can draw graphics on a PDF document.
	CreateTiff	Overloaded. Exports a PDF document to a TIFF image using a stream, the image's largest edge length, and page numbers.
	DeleteAttachment	Returns a value indicating if the attachment is deleted from a PDF document.
	DeleteMarkupAnnotation	Deletes a markup annotation from a page.
	DeleteMarkupAnnotations	Deletes markup annotations specified in the DevExpress.Pdf.PdfMarkupAnnotationData collection.

	DeletePage	Deletes the specified page from the current document.
	Dispose	(Inherited from DevExpress.Pdf.Native.PdfDisposableObject)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Export	Overloaded. Exports interactive form data to the file using the specified form data format.
	FindText	Overloaded. Searches for the specified text in the current document with default parameters.
	FlattenForm	Flattens an entire interactive form.
	FlattenFormField	Flattens a specific form field on an interactive form by its name.
	GetFormData	Returns an object containing values of interactive form data fields.
	GetFormFieldNames	Returns a list of interactive field names in a document.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetImages	Overloaded. Retrieves the images found within the specified document positions using image resolution.
	GetMarkupAnnotationData	Retrieves all text markup annotations from a page in a PDF document.
	GetPageText	Obtains text from the specified page.
	GetText	Overloaded. Retrieves the text found within the specified document positions.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Import	Overloaded. Imports interactive form data from the specified stream.
	InsertNewPage	Inserts a new page with a specified page number and page size into the document.
	LoadDocument	Overloaded. Opens a PDF document from the specified stream.

	NextWord	Returns the next word in a PDF document.
	PrevWord	Returns the previous word in a PDF document.
	Print	Overloaded. Obsolete. Prints the current document, using the specified settings.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveForm	Removes all interactive form fields from a PDF document.
	RemoveFormField	Removes an interactive form field from a document using the field name.
	RenderNewPage	Overloaded. Adds a new page with specified page size and created graphics to a document.
	ResetFormData	Resets all fields of the interactive form to their default values.
	SaveDocument	Overloaded. Saves the current document to the specified file path.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[PdfDocumentProcessor Members](#)[DevExpress.Pdf Namespace](#)[PDF Document API](#)

AddFormFields Method

Adds interactive form fields to a PDF document.

Overload List

Name	Description
void AddFormFields(params PdfAcroFormField[] fields)	Adds interactive form fields to a PDF document.
void AddFormFields(IEnumerable<PdfAcroFormField> fields)	Adds interactive form fields to a PDF document.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.AddFormFields Overload List](#)

Adds interactive form fields to a PDF document.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AddFormFields(  
    ByVal fields As PdfAcroFormField()  
)
```

C#

```
public void AddFormFields(  
    params PdfAcroFormField[] fields  
)
```

Parameters

fields
A DevExpress.Pdf.PdfAcroFormField array containing form fields that should be added to a PDF document.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494642>.

This example shows how to create interactive form fields (e.g., text box and radio button group fields) and add them to a document.

C#

```

(Program.cs)
using DevExpress.Pdf;
namespace AddFormFieldsToExistingDocument {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../\\..\\Document.pdf");
                // Create a text box field specifying the field name, page number, and field location on the page.
                PdfAcroFormTextBoxField textBox = new PdfAcroFormTextBoxField("text box", 1, new PdfRectangle(230, 635, 250, 655));
                // Specify text box text, and appearance.
                textBox.Text = "Text Box";
                textBox.Appearance.BackgroundColor = new PdfRGBColor(0.8, 0.5, 0.3);
                textBox.Appearance.FontSize = 12;
                // Create a radio group field specifying its name and the page number.
                PdfAcroFormRadioGroupField radioGroup = new PdfAcroFormRadioGroupField("Gender Group", 1);
                // Add the first radio button to the group and specify its location using a PdfRectangle object.
                radioGroup.AddButton("button1", new PdfRectangle(230, 635, 250, 655));
                // Add the second radio button to the group.
                radioGroup.AddButton("button2", new PdfRectangle(310, 635, 330, 655));
                // Specify radio group selected index, and appearance.
                radioGroup.SelectedIndex = 0;
                radioGroup.Appearance.BorderAppearance = new PdfAcroFormBorderAppearance() { Color = new PdfRGBColor(0.8, 0.5, 0.3) };
                // Add form fields to the page.
                processor.AddFormFields(textBox, radioGroup);
                // Save the result document.
                processor.SaveDocument("../\\..\\Result.pdf");
            }
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Namespace AddFormFieldsToExistingDocument
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../\\..\\Document.pdf")
                ' Create a text box field specifying the field name, page number, and field location on the page.
                Dim textBox As New PdfAcroFormTextBoxField("text box", 1, New PdfRectangle(230, 690, 280, 710))
                ' Specify text box text, and appearance.
                textBox.Text = "Text Box"
                textBox.Appearance.BackgroundColor = New PdfRGBColor(0.8, 0.5, 0.3)
                textBox.Appearance.FontSize = 12
                ' Create a radio group field specifying its name and the page number.
                Dim radioGroup As New PdfAcroFormRadioGroupField("Gender Group", 1)
                ' Add the first radio button to the group and specify its location using a PdfRectangle object.
                radioGroup.AddButton("button1", New PdfRectangle(230, 635, 250, 655))
                ' Add the second radio button to the group.
                radioGroup.AddButton("button2", New PdfRectangle(310, 635, 330, 655))
                ' Specify radio group selected index, and appearance.
                radioGroup.SelectedIndex = 0
                radioGroup.Appearance.BorderAppearance = New PdfAcroFormBorderAppearance() With {.Color = New PdfRGBColor(0.8, 0.5, 0.3)}
                ' Add form fields to the page.
                processor.AddFormFields(textBox, radioGroup)
                ' Save the result document.
                processor.SaveDocument("../\\..\\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.AddFormFields Overload List](#)

Adds interactive form fields to a PDF document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AddFormFields(  
    ByVal fields As IEnumerable(of PdfAcroFormField)  
)
```

C#

```
public void AddFormFields(  
    IEnumerable<PdfAcroFormField> fields  
)
```

Parameters

fields

A collection of DevExpress.Pdf.PdfAcroFormField objects that represent interactive form fields that should be added to a PDF document.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.AddFormFields Overload List](#)

AddNewPage Method

Adds a new page with the specified page size to the current document.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddNewPage(  
    ByVal mediaBox As PdfRectangle  
) As PdfPage
```

C#

```
public PdfPage AddNewPage(  
    PdfRectangle mediaBox  
)
```

Parameters

mediaBox
A DevExpress.Pdf.PdfRectangle object that is the actual page size.

Return Value

A DevExpress.Pdf.PdfPage object that is the added page.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

The PDF Document Processor allows you to draw graphic primitives on the PDF page, such as lines, Bezier curves, polygons, ellipses, paths, an image and a string.

- This example shows how to draw a business card in the document foreground. To accomplish this task, do the following:
- Create a PDF document processor and add an empty document to it by calling the [CreateEmptyDocument](#) method.
 - Create PDF graphics represented by an instance of the DevExpress.Pdf.PdfGraphics class by calling the [CreateGraphics](#) method. To access DevExpress.Pdf.PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.
 - To draw an image on the PDF page, call the DevExpress.Pdf.PdfGraphics.DrawImage method for the specified image at the specified location with the specified size.
 - To draw text, call the DevExpress.Pdf.PdfGraphics.DrawString method at the specified location with the specified **Brush** and **Font** objects.
 - Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer."
                                );
            }
        }
    }
}
```

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

AddTextMarkupAnnotation Method

Adds a text markup annotation to the specified page area that corresponds to a text that should be annotated on the page.

Overload List

Name	Description
PdfTextMarkupAnnotationData AddTextMarkupAnnotation(PdfDocumentArea area, PdfTextMarkupAnnotationType style)	Adds a text markup annotation to the specified page area that corresponds to a text that should be annotated on the page.
PdfTextMarkupAnnotationData AddTextMarkupAnnotation(int pageNumber, IEnumerable<PdfOrientedRectangle> textRectangles, PdfTextMarkupAnnotationType style)	Adds a text markup annotation to the specified area on the page.
PdfTextMarkupAnnotationData AddTextMarkupAnnotation(int pageNumber, PdfRectangle rectangle, PdfTextMarkupAnnotationType style)	Adds a text markup annotation to the specified rectangle that corresponds to a text that should be annotated on the page.
PdfTextMarkupAnnotationData AddTextMarkupAnnotation(PdfDocumentPosition startPosition, PdfDocumentPosition endPosition, PdfTextMarkupAnnotationType style)	Adds a text markup annotation for a text located within the specified positions on the page.
PdfTextMarkupAnnotationData AddTextMarkupAnnotation(int pageNumber, PdfOrientedRectangle textRectangle, PdfTextMarkupAnnotationType style)	Adds a text markup annotation to the specified area on the page.
PdfTextMarkupAnnotationData AddTextMarkupAnnotation(int pageNumber, IEnumerable<PdfQuadrilateral> quads, PdfTextMarkupAnnotationType style)	Adds a text markup annotation to the specified area on the page.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.AddTextMarkupAnnotation Overload List](#)

Adds a text markup annotation to the specified page area that corresponds to a text that should be annotated on the page.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddTextMarkupAnnotation(  
    ByVal area As PdfDocumentArea,  
    ByVal style As PdfTextMarkupAnnotationType  
) As PdfTextMarkupAnnotationData
```

C#

```
public PdfTextMarkupAnnotationData AddTextMarkupAnnotation(  
    PdfDocumentArea area,  
    PdfTextMarkupAnnotationType style  
)
```

Parameters

area
A DevExpress.Pdf.PdfDocumentArea object that specifies the document area that should be associated with a text for which the markup annotation is added.

style
A DevExpress.Pdf.PdfTextMarkupAnnotationType enumeration value that specifies the markup annotation type to be added to a page.

Return Value

A DevExpress.Pdf.PdfTextMarkupAnnotationData object that represents the created text markup annotation on a page.

Remarks

After creating a text markup annotation, you can specify the markup annotation properties using DevExpress.Pdf.PdfTextMarkupAnnotationData object.

if a specified page area does not correspond to a text on the page, the **AddTextMarkupAnnotation** method returns **null**.

Example

<a>Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T545395 .	

This example shows how to create a text markup annotation that highlights a text in a PDF document and specify the annotation properties.

To add a text markup annotation to a page, call one of [AddTextMarkupAnnotation](#) overload methods, where specify the page number and a page area corresponding to a text that should be annotated on this page. Note that if a specified page area does not correspond to a text on the page, the annotation is not created and [AddTextMarkupAnnotation](#) overload methods return **null**.

C#	
<pre>(Program.cs) using System; using DevExpress.Pdf; namespace CreateMarkupAnnotation { class Program { static void Main(string[] args) { // Create a PDF Document Processor. using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) { // Load a document. processor.LoadDocument("../\\..\\Document.pdf"); // Add a text markup annotation to the first page for a text corresponding to the specified PdfTextMarkupAnnotationData annot = processor.AddTextMarkupAnnotation(1, new PdfRectangle(PdfTextMarkupAnnotationType.Highlight); if (annot != null) { // Specify the annotation properties. annot.Author = "Bill Smith"; annot.Contents = "Note"; annot.Color = new PdfRGBColor(0.8, 0.2, 0.1); // Save the result document. processor.SaveDocument("../\\..\\Result.pdf"); } else Console.WriteLine("The annotation cannot be added to a page. Make sure, a specified page } } } }</pre>	

Visual Basic	
--------------	--

```

(Program.vb)
Imports System
Imports DevExpress.Pdf
Namespace CreateMarkupAnnotation
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Add a text markup annotation to the first page for a text corresponding to the specified
                Dim annot As PdfTextMarkupAnnotationData = processor.AddTextMarkupAnnotation(1, New PdfRect
                If annot IsNot Nothing Then
                    ' Specify the annotation properties.
                    annot.Author = "Bill Smith"
                    annot.Contents = "Note"
                    annot.Color = New PdfRGBColor(0.8, 0.2, 0.1)
                    ' Save the result document.
                    processor.SaveDocument("../..\Result.pdf")
                Else
                    Console.WriteLine("The annotation cannot be added to a page. Make sure, a specified pa
                End If
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.AddTextMarkupAnnotation Overload List](#)

Adds a text markup annotation to the specified area on the page.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Function AddTextMarkupAnnotation(
    ByVal pageNumber As Integer,
    ByVal textRectangles As IEnumerable(of PdfOrientedRectangle),
    ByVal style As PdfTextMarkupAnnotationType
) As PdfTextMarkupAnnotationData

```

C#

```

public PdfTextMarkupAnnotationData AddTextMarkupAnnotation(
    int pageNumber,
    IEnumerable<PdfOrientedRectangle> textRectangles,
    PdfTextMarkupAnnotationType style
)

```

Parameters

pageNumber

An integer value that specifies the number of the page where the annotation should be added.

textRectangles

A collection of DevExpress.Pdf.PdfOrientedRectangle objects that represent rectangles specifying a page area where the annotation will be added.

style

A DevExpress.Pdf.PdfTextMarkupAnnotationType enumeration value that specifies the markup annotation type to be added to a page.

Return Value

A DevExpress.Pdf.PdfTextMarkupAnnotationData object that represents the created text markup annotation on a page.

Remarks

After creating a text markup annotation, you can specify the markup annotation using properties of the `DevExpress.Pdf.PdfTextMarkupAnnotationData` class.

This method creates a text markup annotation for the specified page area. This method returns **null** if the collection of `DevExpress.Pdf.PdfOrientedRectangle` objects is empty.

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.AddTextMarkupAnnotation Overload List](#)

Adds a text markup annotation to the specified rectangle that corresponds to a text that should be annotated on the page.

Namespace: [DevExpress.Pdf](#)
Assembly: `DevExpress.Docs.v18.1.dll`

Syntax

Visual Basic

```
Public Function AddTextMarkupAnnotation(  
    ByVal pageNumber As Integer,  
    ByVal rectangle As PdfRectangle,  
    ByVal style As PdfTextMarkupAnnotationType  
) As PdfTextMarkupAnnotationData
```

C#

```
public PdfTextMarkupAnnotationData AddTextMarkupAnnotation(  
    int pageNumber,  
    PdfRectangle rectangle,  
    PdfTextMarkupAnnotationType style  
)
```

Parameters

pageNumber
An integer value that specifies the number of a page where the annotation should be added.

rectangle
A `DevExpress.Pdf.PdfRectangle` object that represents the rectangle corresponding to a text that should be annotated on the page.

style
A `DevExpress.Pdf.PdfTextMarkupAnnotationType` enumeration value that specifies the markup annotation type to be added to a page.

Return Value

A `DevExpress.Pdf.PdfTextMarkupAnnotationData` object that represents the created text markup annotation on a page.

Remarks

After creating a text markup annotation, you can specify the markup annotation properties using `DevExpress.Pdf.PdfTextMarkupAnnotationData` object.

if a specified rectangle does not correspond to a text on the page, the annotation is not created and the method returns **null**.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T545395>.

This example shows how to create a text markup annotation that highlights a text in a PDF document and specify the annotation properties.

To add a text markup annotation to a page, call one of [AddTextMarkupAnnotation](#) overload methods, where specify the page number and a page area corresponding to a text that should be annotated on this page. Note that if a specified page area does not correspond to a text on the page, the annotation is not created and [AddTextMarkupAnnotation](#) overload methods return **null**.

C#

```
(Program.cs)
using System;
using DevExpress.Pdf;
namespace CreateMarkupAnnotation {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\Document.pdf");
                // Add a text markup annotation to the first page for a text corresponding to the specified
                PdfTextMarkupAnnotationData annot = processor.AddTextMarkupAnnotation(1, new PdfRectangle
                    PdfTextMarkupAnnotationType.Highlight);

                if (annot != null) {
                    // Specify the annotation properties.
                    annot.Author = "Bill Smith";
                    annot.Contents = "Note";
                    annot.Color = new PdfRGBColor(0.8, 0.2, 0.1);
                    // Save the result document.
                    processor.SaveDocument("..\\..\\Result.pdf");
                }
                else
                    Console.WriteLine("The annotation cannot be added to a page. Make sure, a specified page
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports System
Imports DevExpress.Pdf
Namespace CreateMarkupAnnotation
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("..\\..\\Document.pdf")
                ' Add a text markup annotation to the first page for a text corresponding to the specified
                Dim annot As PdfTextMarkupAnnotationData = processor.AddTextMarkupAnnotation(1, New PdfRect
                If annot IsNot Nothing Then
                    ' Specify the annotation properties.
                    annot.Author = "Bill Smith"
                    annot.Contents = "Note"
                    annot.Color = New PdfRGBColor(0.8, 0.2, 0.1)
                    ' Save the result document.
                    processor.SaveDocument("..\\..\\Result.pdf")
                Else
                    Console.WriteLine("The annotation cannot be added to a page. Make sure, a specified page
                End If
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.AddTextMarkupAnnotation Overload List](#)

Adds a text markup annotation for a text located within the specified positions on the page.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddTextMarkupAnnotation(  
    ByVal startPosition As PdfDocumentPosition,  
    ByVal endPosition As PdfDocumentPosition,  
    ByVal style As PdfTextMarkupAnnotationType  
) As PdfTextMarkupAnnotationData
```

C#

```
public PdfTextMarkupAnnotationData AddTextMarkupAnnotation(  
    PdfDocumentPosition startPosition,  
    PdfDocumentPosition endPosition,  
    PdfTextMarkupAnnotationType style  
)
```

Parameters

startPosition

A DevExpress.Pdf.PdfDocumentPosition object that represents the start point of a page area that corresponds to a text that should be annotated on the page.

endPosition

A DevExpress.Pdf.PdfDocumentPosition object that represents the end point of a page area that corresponds to a text that should be annotated on the page.

style

A DevExpress.Pdf.PdfTextMarkupAnnotationType enumeration value that specifies the markup annotation type to be added to a page.

Return Value

A DevExpress.Pdf.PdfTextMarkupAnnotationData object that represents the created text markup annotation on a page.

Remarks

If a specified page area does not correspond to a text on the page, the annotation is not created and the method returns **null**.

Note

If a text for which the markup annotation is created, starts on one page and ends on another page, this method adds annotation only to a page where the text starts.

After creating a text markup annotation, you can specify the markup annotation properties using DevExpress.Pdf.PdfTextMarkupAnnotationData object.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T545395>.

This example shows how to create a text markup annotation that highlights a text in a PDF document and specify the annotation properties.

To add a text markup annotation to a page, call one of [AddTextMarkupAnnotation](#) overload methods, where specify the page number and a page area corresponding to a text that should be annotated on this page. Note that if a specified page area does not correspond to a text on the page, the annotation is not created and [AddTextMarkupAnnotation](#) overload methods return **null**.

C#

```

(Program.cs)
using System;
using DevExpress.Pdf;
namespace CreateMarkupAnnotation {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\Document.pdf");
                // Add a text markup annotation to the first page for a text corresponding to the specified
                PdfTextMarkupAnnotationData annot = processor.AddTextMarkupAnnotation(1, new PdfRectangle
                    PdfTextMarkupAnnotationType.Highlight);

                if (annot != null) {
                    // Specify the annotation properties.
                    annot.Author = "Bill Smith";
                    annot.Contents = "Note";
                    annot.Color = new PdfRGBColor(0.8, 0.2, 0.1);
                    // Save the result document.
                    processor.SaveDocument("..\\..\\Result.pdf");
                }
                else
                    Console.WriteLine("The annotation cannot be added to a page. Make sure, a specified page
            }
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports System
Imports DevExpress.Pdf
Namespace CreateMarkupAnnotation
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("..\\..\\Document.pdf")
                ' Add a text markup annotation to the first page for a text corresponding to the specified
                Dim annot As PdfTextMarkupAnnotationData = processor.AddTextMarkupAnnotation(1, New PdfRect
                If annot IsNot Nothing Then
                    ' Specify the annotation properties.
                    annot.Author = "Bill Smith"
                    annot.Contents = "Note"
                    annot.Color = New PdfRGBColor(0.8, 0.2, 0.1)
                    ' Save the result document.
                    processor.SaveDocument("..\\..\\Result.pdf")
                Else
                    Console.WriteLine("The annotation cannot be added to a page. Make sure, a specified page
                End If
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.AddTextMarkupAnnotation Overload List](#)

Adds a text markup annotation to the specified area on the page.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddTextMarkupAnnotation(  
    ByVal pageNumber As Integer,  
    ByVal textRectangle As PdfOrientedRectangle,  
    ByVal style As PdfTextMarkupAnnotationType  
) As PdfTextMarkupAnnotationData
```

C#

```
public PdfTextMarkupAnnotationData AddTextMarkupAnnotation(  
    int pageNumber,  
    PdfOrientedRectangle textRectangle,  
    PdfTextMarkupAnnotationType style  
)
```

Parameters

pageNumber

An integer value that specifies the number of the page where the annotation should be added.

textRectangle

A DevExpress.Pdf.PdfOrientedRectangle object that represents a rectangle specifying a page area where the annotation will be added.

style

A DevExpress.Pdf.PdfTextMarkupAnnotationType enumeration value that specifies the markup annotation type to be added to a page.

Return Value

A DevExpress.Pdf.PdfTextMarkupAnnotationData object that represents the created text markup annotation on a page.

Remarks

After creating a text markup annotation, you can specify the markup annotation properties using a DevExpress.Pdf.PdfTextMarkupAnnotationData object.

if a specified page area does not correspond to a text on the page, the **AddTextMarkupAnnotation** method returns **null**.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.AddTextMarkupAnnotation Overload List](#)

Adds a text markup annotation to the specified area on the page.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function AddTextMarkupAnnotation(  
    ByVal pageNumber As Integer,  
    ByVal quads As IEnumerable(of PdfQuadrilateral),  
    ByVal style As PdfTextMarkupAnnotationType  
) As PdfTextMarkupAnnotationData
```

C#

```
public PdfTextMarkupAnnotationData AddTextMarkupAnnotation(  
    int pageNumber,  
    IEnumerable<PdfQuadrilateral> quads,  
    PdfTextMarkupAnnotationType style  
)
```

Parameters

pageNumber

An integer value that specifies the number of the page where the annotation should be added.

quads

A collection of `DevExpress.Pdf.PdfQuadrilateral` objects that represent quadrilaterals specifying the text markup annotation bounds on a page.

style

A `DevExpress.Pdf.PdfTextMarkupAnnotationType` enumeration value that specifies the markup annotation type to be added to a page.

Return Value

A `DevExpress.Pdf.PdfTextMarkupAnnotationData` object that represents the created text markup annotation on a page.

Remarks

After creating a text markup annotation, you can specify the markup annotation using properties of the `DevExpress.Pdf.PdfTextMarkupAnnotationData` class.

This method creates a text markup annotation for the specified quadrilaterals. This method returns **null** if the collection of `DevExpress.Pdf.PdfQuadrilateral` objects is empty.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.AddTextMarkupAnnotation Overload List](#)

AppendDocument Method

Appends a PDF document located at the specified file stream to the end of the current document starting from the new page.

Overload List

Name	Description
void AppendDocument(Stream stream)	Appends a PDF document located at the specified file stream to the end of the current document starting from the new page.
void AppendDocument(string path)	Appends a PDF document located at the specified path to the end of the current document starting from the new page.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.AppendDocument Overload List](#)

Appends a PDF document located at the specified file stream to the end of the current document starting from the new page.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AppendDocument(  
    ByVal stream As Stream  
)
```

C#

```
public void AppendDocument(  
    Stream stream  
)
```

Parameters

stream

A [System.IO.Stream](#) value, specifying the location of the appended document.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.AppendDocument Overload List](#)

Appends a PDF document located at the specified path to the end of the current document starting from the new page.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AppendDocument(  
    ByVal path As String  
)
```

C#

```
public void AppendDocument(  
    string path  
)
```

Parameters

path

A [System.String](#) value, specifying the location of the appended document.

See Also[PdfDocumentProcessor Class](#)[PdfDocumentProcessor Members](#)[DevExpress.Pdf Namespace](#)[PdfDocumentProcessor.AppendDocument Overload List](#)

ApplyFormData Method

Fills an interactive form with data.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ApplyFormData(  
    ByVal data As PdfFormData  
)
```

C#

```
public void ApplyFormData(  
    PdfFormData data  
)
```

Parameters

data

A DevExpress.Pdf.PdfFormData object containing data to be applied to a form.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the DevExpress.Pdf.PdfFormData.Value property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#

```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic

```
(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

AttachFile Method

Attaches a file to the PDF document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AttachFile(  
    ByVal attachment As PdfFileAttachment  
)
```

C#

```
public void AttachFile(  
    PdfFileAttachment attachment  
)
```

Parameters

attachment

A DevExpress.Pdf.PdfFileAttachment object that contains settings to attach a file to the PDF document.

Remarks

Use this method to include an attachment to the PDF document using the necessary settings such as attachment data (DevExpress.Pdf.PdfFileAttachment.Data), a file name (DevExpress.Pdf.PdfFileAttachment.FileName), creation date (DevExpress.Pdf.PdfFileAttachment.CreationDate), modification date (DevExpress.Pdf.PdfFileAttachment.ModificationDate) and other settings.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T304234>.

This example shows how to programmatically attach a file to the PDF document.

To do this:

- Create a DevExpress.Pdf.PdfFileAttachment object;
- Specify the attachment creation date, description, and file name using DevExpress.Pdf.PdfFileAttachment.CreationDate, DevExpress.Pdf.PdfFileAttachment.Description, and DevExpress.Pdf.PdfFileAttachment.FileName properties. To specify the data for the attachment, use the DevExpress.Pdf.PdfFileAttachment.Data property;
- If required, you can specify additional properties of an attached file, for example, the file's mime type and relationship using DevExpress.Pdf.PdfFileAttachment.MimeType and DevExpress.Pdf.PdfFileAttachment.Relationship properties, respectively.
- Call the [AttachFile](#) method with a file attachment object used as a parameter.
- Save the attachment to a document by calling the [SaveDocument](#) method.

C#

```

(Program.cs)
using DevExpress.Pdf;
using System;
using System.IO;
namespace AttachFile {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\Document.pdf");
                // Attach a file to the PDF document.
                processor.AttachFile(new PdfFileAttachment() {
                    CreationDate = DateTime.Now,
                    Description = "This is my attach file.",
                    FileName = "MyAttach.txt",
                    Data = File.ReadAllBytes("..\\..\\FileToAttach.txt")
                });
                // The attached document.
                processor.SaveDocument("..\\..\\Result.pdf");
            }
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.IO
Namespace AttachFile
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("..\\..\\Document.pdf")
                ' Attach a file to the PDF document.
                processor.AttachFile(New PdfFileAttachment() With { _
                    .CreationDate = Date.Now, _
                    .Description = "This is my attach file.", _
                    .FileName = "MyAttach.txt", _
                    .Data = File.ReadAllBytes("..\\..\\FileToAttach.txt") _
                })
                ' The attached document.
                processor.SaveDocument("..\\..\\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)

CheckFormFieldNameCollisions Method

Checks interactive form fields to find a collision in the form field names.

Overload List

Name	Description
IList<PdfAcroFormFieldNameCollision> CheckFormFieldNameCollisions(params PdfAcroFormField[] fields)	Checks interactive form fields to find a collision in the form field names.
IList<PdfAcroFormFieldNameCollision> CheckFormFieldNameCollisions(IEnumerable<PdfAcroFormFie<ld> fields)	Checks interactive form fields to find a collision in the form field names.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CheckFormFieldNameCollisions Overload List](#)

Checks interactive form fields to find a collision in the form field names.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CheckFormFieldNameCollisions(  
    ByVal fields As PdfAcroFormField()  
) As IList(of PdfAcroFormFieldNameCollision)
```

```
C#  
public IList<PdfAcroFormFieldNameCollision> CheckFormFieldNameCollisions(  
    params PdfAcroFormField[] fields  
)
```

Parameters

fields
A DevExpress.Pdf.PdfAcroFormField array containing interactive form fields that should be checked to find a collision in the form field names.

Return Value

A collection of DevExpress.Pdf.PdfAcroFormFieldNameCollision objects that contains information about a collision found in interactive form field names.

Remarks

Form field names must be unique within an interactive form.

To obtain the interactive form field in which a collision was found with a field name, use the DevExpress.Pdf.PdfAcroFormFieldNameCollision.Field property.

You can also get the forbidden field names using the DevExpress.Pdf.PdfAcroFormFieldNameCollision.ForbiddenNames property.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CheckFormFieldNameCollisions Overload List](#)

Checks interactive form fields to find a collision in the form field names.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function CheckFormFieldNameCollisions(  
    ByVal fields As IEnumerable(of PdfAcroFormField)  
) As IList(of PdfAcroFormFieldNameCollision)
```

C#

```
public IList<PdfAcroFormFieldNameCollision> CheckFormFieldNameCollisions(  
    IEnumerable<PdfAcroFormField> fields  
)
```

Parameters*fields*

A list of DevExpress.Pdf.PdfAcroFormField objects containing interactive form fields that should be checked to find a collision in the form field names.

Return Value

A collection of DevExpress.Pdf.PdfAcroFormFieldNameCollision objects that contains information about a collision found in interactive form field names.

Remarks

Form field names must be unique within an interactive form.

To obtain the interactive form field in which a collision was found with a field name, use the DevExpress.Pdf.PdfAcroFormFieldNameCollision.Field property.

You can also get the forbidden field names using the DevExpress.Pdf.PdfAcroFormFieldNameCollision.ForbiddenNames property.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CheckFormFieldNameCollisions Overload List](#)

CloseDocument Method

Closes the current document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub CloseDocument()
```

C#

```
public void CloseDocument()
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

CreateBitmap Method

Exports a PDF page to a bitmap image.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CreateBitmap(  
    ByVal pageNumber As Integer,  
    ByVal largestEdgeLength As Integer  
) As Bitmap
```

C#

```
public Bitmap CreateBitmap(  
    int pageNumber,  
    int largestEdgeLength  
)
```

Parameters


pageNumber
An integer value, specifying the converted page's number.
largestEdgeLength
An integer value, specifying the length of the image's largest dimension, in pixels.

Return Value

A [System.Drawing.Bitmap](#) object.

Remarks

The **CreateBitmap** method has two parameters: the converted page's number and the length of the image's largest dimension in pixels. The latter parameter determines the output image height for pages in the portrait orientation and width - for landscape pages.

 **Azure Compatibility**

The **CreateBitmap** method works on Azure Roles (Worker and/or Web) and does not work on Azure web site.

Example

The following example illustrates how to convert pages to bitmap images.

 **Show Me**

The complete sample project is available at <https://github.com/DevExpress-Examples/how-to-export-a-pdf-document-to-multi-page-tiff-and-bitmap>

C#

```

using DevExpress.Pdf;
using System.Drawing;
namespace ExportToBitmap {
    class Program {
        static void Main(string[] args) {
            int largestEdgeLength = 1000;
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\Document.pdf");
                for (int i = 1; i <= processor.Document.Pages.Count; i++) {
                    // Export pages to bitmaps.
                    Bitmap image = processor.CreateBitmap(i, largestEdgeLength);
                    // Save the bitmaps.
                    image.Save("..\\..\\MyBitmap" + i + ".bmp");
                }
            }
        }
    }
}

```

Visual Basic

```

Imports DevExpress.Pdf
Imports System.Drawing
Namespace ExportToBitmap
    Class Program
        Private Shared Sub Main(ByVal args As String())
            Dim largestEdgeLength As Integer = 1000
            Using processor As PdfDocumentProcessor = New PdfDocumentProcessor()
                processor.LoadDocument("..\\..\\Document.pdf")
                For i As Integer = 1 To processor.Document.Pages.Count
                    Dim image As Bitmap = processor.CreateBitmap(i, largestEdgeLength)
                    image.Save("..\\..\\MyBitmap" & i & ".bmp")
                Next
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)

CreateDestination Method

Creates a destination for targets in the document (e.g., bookmarks) using a page number.

Overload List

Name	Description
PdfDestination CreateDestination(int pageNumber)	Creates a destination for targets in the document (e.g., bookmarks) using a page number.
PdfDestination CreateDestination(int pageNumber, Single zoomFactor)	Creates a destination for targets in the document (e.g., bookmarks) using a page number, and zoom factor.
PdfDestination CreateDestination(int pageNumber, Single x, Single y)	Creates a destination for targets in the document (e.g., bookmarks) using a page number, and page coordinates.
PdfDestination CreateDestination(int pageNumber, Single x, Single y, Single zoomFactor)	Creates a destination for targets in the document (e.g., bookmarks) using a page number, page coordinates, and zoom factor.
PdfDestination CreateDestination(int pageNumber, Single x, Single y, Single dpiX, Single dpiY)	Creates a destination for targets in the document (e.g., bookmarks) using a page number, page coordinates, and DPI.
PdfDestination CreateDestination(int pageNumber, Single x, Single y, Single dpiX, Single dpiY, Single zoomFactor)	Creates a destination for targets in the document (e.g., bookmarks) using a page number, page coordinates, DPI and zoom factor.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CreateDestination Overload List](#)

Creates a destination for targets in the document (e.g., bookmarks) using a page number.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CreateDestination(  
    ByVal pageNumber As Integer  
) As PdfDestination
```

C#

```
public PdfDestination CreateDestination(  
    int pageNumber  
)
```

Parameters

pageNumber
An integer value that is the page number where the destination is created.

Return Value

A DevExpress.Pdf.PdfDestination object that is the page destination.

Remarks

The overloaded **CreateDestination** method converts world coordinates to page coordinates. See the [Coordinate Systems](#) topic to learn more.

Use this method to create a destination that points to a specific location and view in a document. The created destination is assigned to the DevExpress.Pdf.PdfBookmark.Destination property.

For more information about the bookmarks creation, see the [Bookmarks](#) topic.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CreateDestination Overload List](#)

Creates a destination for targets in the document (e.g., bookmarks) using a page number, and zoom factor.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic

```
Public Function CreateDestination(  
    ByVal pageNumber As Integer,  
    ByVal zoomFactor As Single  
) As PdfDestination
```

C#

```
public PdfDestination CreateDestination(  
    int pageNumber,  
    Single zoomFactor  
)
```

Parameters
pageNumber
An integer value that is the page number where the destination is created.
zoomFactor
The zoom level by which a page destination is created.

Return Value
A DevExpress.Pdf.PdfDestination object that is the page destination.

Remarks
The overloaded **CreateDestination** method converts world coordinates to page coordinates. See the [Coordinate Systems](#) topic to learn more.
Use this method to create a destination that points to a specific location and view in a document. The created destination is assigned to the DevExpress.Pdf.PdfBookmark.Destination property.
For more information about the bookmarks creation, see the [Bookmarks](#) topic.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T495077>.

This example shows how to create bookmarks in code.

- To do this:
- Create a DevExpress.Pdf.PdfBookmark object;
 - Specify the bookmark title and destination using the DevExpress.Pdf.PdfBookmark.Title and DevExpress.Pdf.PdfBookmark.Destination properties. To create a destination, call one of the [CreateDestination](#) overloaded methods.
 - Add the bookmark to the PdfBookmark collection, which is accessed from the DevExpress.Pdf.PdfDocument.Bookmarks property.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace AddBookmarks {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\Document.pdf");
                // Create bookmarks and add them to the PDF document.
                PdfDestination destination1 = processor.CreateDestination(1, 180, 150);
                PdfDestination destination2 = processor.CreateDestination(1, 168, 230);
                PdfDestination destination3 = processor.CreateDestination(1, 20, 350);
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "PDF Document Processor", Destination = destination1 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Display, Print and Export PDF", Destination = destination2 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Learn More", Destination = destination3 });
                // Save the result document.
                processor.SaveDocument("..\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace AddBookmarks
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("..\\..\\Document.pdf")
                ' Create bookmarks and add them to the PDF document.
                Dim destination1 As PdfDestination = processor.CreateDestination(1, 180, 150)
                Dim destination2 As PdfDestination = processor.CreateDestination(1, 168, 230)
                Dim destination3 As PdfDestination = processor.CreateDestination(1, 20, 350)
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "PDF Document Processor", .Destination = destination1})
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "Display, Print and Export PDF", .Destination = destination2})
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "Learn More", .Destination = destination3})
                ' Save the result document.
                processor.SaveDocument("..\\..\\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateDestination Overload List](#)

Creates a destination for targets in the document (e.g., bookmarks) using a page number, and page coordinates.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CreateDestination(
    ByVal pageNumber As Integer,
    ByVal x As Single,
    ByVal y As Single
) As PdfDestination
```

C#

```
public PdfDestination CreateDestination(  
    int pageNumber,  
    Single x,  
    Single y  
)
```

Parameters

pageNumber

An integer value that is the page number where the destination is created.

x

A [System.Single](#) object that is the horizontal page coordinate value.

y

A [System.Single](#) object that is the vertical page coordinate value.

Return Value

A DevExpress.Pdf.PdfDestination object that is the page destination.

Remarks

The overloaded **CreateDestination** method converts world coordinates to page coordinates. See the [Coordinate Systems](#) topic to learn more.

Use this method to create a destination that points to a specific location and view in a document. The created destination is assigned to the DevExpress.Pdf.PdfBookmark.Destination property.

For more information about the bookmarks creation, see the [Bookmarks](#) topic.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateDestination Overload List](#)

Creates a destination for targets in the document (e.g., bookmarks) using a page number, page coordinates, and zoom factor.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function CreateDestination(  
    ByVal pageNumber As Integer,  
    ByVal x As Single,  
    ByVal y As Single,  
    ByVal zoomFactor As Single  
) As PdfDestination
```

C#

```
public PdfDestination CreateDestination(  
    int pageNumber,  
    Single x,  
    Single y,  
    Single zoomFactor  
)
```

Parameters

pageNumber

An integer value that is the page number where the destination is created.

x

A [System.Single](#) object that is the horizontal page coordinate value.

y

A [System.Single](#) object that is the vertical page coordinate value.

zoomFactor

The zoom level by which a page destination is created.

Return Value

A DevExpress.Pdf.PdfDestination object that is the page destination.

Remarks

The overloaded **CreateDestination** method converts world coordinates to page coordinates. See the [Coordinate Systems](#) topic to learn more.

Use this method to create a destination that points to a specific location and view in a document. The created destination is assigned to the `DevExpress.Pdf.PdfBookmark.Destination` property.

For more information about the bookmarks creation, see the [Bookmarks](#) topic.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T495077>.

This example shows how to create bookmarks in code.

To do this:

- Create a `DevExpress.Pdf.PdfBookmark` object;
- Specify the bookmark title and destination using the `DevExpress.Pdf.PdfBookmark.Title` and `DevExpress.Pdf.PdfBookmark.Destination` properties. To create a destination, call one of the [CreateDestination](#) overloaded methods.
- Add the bookmark to the `PdfBookmark` collection, which is accessed from the `DevExpress.Pdf.PdfDocument.Bookmarks` property.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace AddBookmarks {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../\\..\\Document.pdf");
                // Create bookmarks and add them to the PDF document.
                PdfDestination destination1 = processor.CreateDestination(1, 180, 150);
                PdfDestination destination2 = processor.CreateDestination(1, 168, 230);
                PdfDestination destination3 = processor.CreateDestination(1, 20, 350);
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "PDF Document Processor", Destination = destination1 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Display, Print and Export PDF", Destination = destination2 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Learn More", Destination = destination3 });
                // Save the result document.
                processor.SaveDocument("../\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Namespace AddBookmarks
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Create bookmarks and add them to the PDF document.
                Dim destination1 As PdfDestination = processor.CreateDestination(1, 180, 150)
                Dim destination2 As PdfDestination = processor.CreateDestination(1, 168, 230)
                Dim destination3 As PdfDestination = processor.CreateDestination(1, 20, 350)
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "PDF Document Processor"})
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "Display, Print and Export"})
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "Learn More", .Destination = destination3})
                ' Save the result document.
                processor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateDestination Overload List](#)

Creates a destination for targets in the document (e.g., bookmarks) using a page number, page coordinates, and DPI.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Function CreateDestination(
    ByVal pageNumber As Integer,
    ByVal x As Single,
    ByVal y As Single,
    ByVal dpiX As Single,
    ByVal dpiY As Single
) As PdfDestination

```

C#

```

public PdfDestination CreateDestination(
    int pageNumber,
    Single x,
    Single y,
    Single dpiX,
    Single dpiY
)

```

Parameters

pageNumber

An integer value that is the page number where the destination is created.

x

A [System.Single](#) object that is the horizontal page coordinate value.

y

A [System.Single](#) object that is the vertical page coordinate value.

dpiX

A [System.Single](#) object that is the value, in dots per inch, for the horizontal resolution.

dpiY

A [System.Single](#) object that is the value, in dots per inch, for the vertical resolution.

Return Value

A DevExpress.Pdf.PdfDestination object that is the page destination.

Remarks

The overloaded **CreateDestination** method converts world coordinates to page coordinates. See the [Coordinate Systems](#) topic to learn more.

Use this method to create a destination that points to a specific location and view in a document. The created destination is assigned to the DevExpress.Pdf.PdfBookmark.Destination property.

For more information about the bookmarks creation, see the [Bookmarks](#) topic.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateDestination Overload List](#)

Creates a destination for targets in the document (e.g., bookmarks) using a page number, page coordinates, DPI and zoom factor.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CreateDestination(  
    ByVal pageNumber As Integer,  
    ByVal x As Single,  
    ByVal y As Single,  
    ByVal dpiX As Single,  
    ByVal dpiY As Single,  
    ByVal zoomFactor As Single  
) As PdfDestination
```

C#

```
public PdfDestination CreateDestination(  
    int pageNumber,  
    Single x,  
    Single y,  
    Single dpiX,  
    Single dpiY,  
    Single zoomFactor  
)
```

Parameters

pageNumber

An integer value that is the page number where the destination is created.

x

A [System.Single](#) object that is the horizontal page coordinate value.

y

A [System.Single](#) object that is the vertical page coordinate value.

dpiX

A [System.Single](#) object that is the value, in dots per inch, for the horizontal resolution.

dpiY

A [System.Single](#) object that is the value, in dots per inch, for the vertical resolution.

zoomFactor

The zoom level by which a page destination is created.

Return Value

A DevExpress.Pdf.PdfDestination object that is the page destination.

Remarks

The overloaded **CreateDestination** method converts world coordinates to page coordinates. See the [Coordinate Systems](#) topic to learn more.

Use this method to create a destination that points to a specific location and view in a document. The created destination is assigned to the `DevExpress.Pdf.PdfBookmark.Destination` property.

For more information about the bookmarks creation, see the [Bookmarks](#) topic.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T495077>.

This example shows how to create bookmarks in code.

To do this:

- Create a `DevExpress.Pdf.PdfBookmark` object;
- Specify the bookmark title and destination using the `DevExpress.Pdf.PdfBookmark.Title` and `DevExpress.Pdf.PdfBookmark.Destination` properties. To create a destination, call one of the [CreateDestination](#) overloaded methods.
- Add the bookmark to the `PdfBookmark` collection, which is accessed from the `DevExpress.Pdf.PdfDocument.Bookmarks` property.

C#

```
(Program.cs)
using DevExpress.Pdf;
namespace AddBookmarks {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\..\\Document.pdf");
                // Create bookmarks and add them to the PDF document.
                PdfDestination destination1 = processor.CreateDestination(1, 180, 150);
                PdfDestination destination2 = processor.CreateDestination(1, 168, 230);
                PdfDestination destination3 = processor.CreateDestination(1, 20, 350);
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "PDF Document Processor", Destination = destination1 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Display, Print and Export PDF", Destination = destination2 });
                processor.Document.Bookmarks.Add(new PdfBookmark() { Title = "Learn More", Destination = destination3 });
                // Save the result document.
                processor.SaveDocument("..\\..\\..\\Result.pdf");
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Namespace AddBookmarks
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Create bookmarks and add them to the PDF document.
                Dim destination1 As PdfDestination = processor.CreateDestination(1, 180, 150)
                Dim destination2 As PdfDestination = processor.CreateDestination(1, 168, 230)
                Dim destination3 As PdfDestination = processor.CreateDestination(1, 20, 350)
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "PDF Document Processor", .Destination = destination1})
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "Display, Print and Export", .Destination = destination2})
                processor.Document.Bookmarks.Add(New PdfBookmark() With {.Title = "Learn More", .Destination = destination3})
                ' Save the result document.
                processor.SaveDocument("../..\Result.pdf")
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateDestination Overload List](#)

CreateEmptyDocument Method

Creates an empty PDF document with no pages.

Overload List

Name	Description
void CreateEmptyDocument()	Creates an empty PDF document with no pages.
void CreateEmptyDocument(string path)	Creates an empty PDF document with no pages using the file path.
void CreateEmptyDocument(Stream stream)	Creates an empty PDF document with no pages using a stream.
void CreateEmptyDocument(Stream stream, PdfSaveOptions saveOptions)	Creates an empty PDF document with no pages using the stream and PDF save options (containing encryption settings and a signature) that will be applied to the document when it is saved.
void CreateEmptyDocument(string path, PdfSaveOptions saveOptions)	Creates an empty PDF document with no pages using the file path and PDF save options (containing encryption settings and a signature) that will be applied to the document when it is saved.
void CreateEmptyDocument(Stream stream, PdfCreationOptions creationOptions)	Creates an empty PDF with no pages using a stream and PDF creation options.
void CreateEmptyDocument(string path, PdfCreationOptions creationOptions)	Creates an empty PDF document with no pages using the file path and PDF creation options.
void CreateEmptyDocument(Stream stream, PdfSaveOptions saveOptions, PdfCreationOptions creationOptions)	Creates an empty PDF document with no pages using a stream, PDF save options (containing encryption settings and a signature) that will be applied to the document when it is saved, and PDF creation options.
void CreateEmptyDocument(string path, PdfSaveOptions saveOptions, PdfCreationOptions creationOptions)	Creates an empty PDF document with no pages using the file path, PDF save options (containing encryption settings and a signature) that will be applied to the document when it is saved, and PDF creation options.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF document with no pages.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

```
Visual Basic
Public Sub CreateEmptyDocument()

C#
public void CreateEmptyDocument()
```

Remarks

Use this method to create an empty document from scratch (instead of merging two existing documents). Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Generating a Document](#) topic.

 **Note**

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

The PDF Document Processor allows you to draw graphic primitives on the PDF page, such as lines, Bezier curves, polygons, ellipses, paths, an image and a string.

- This example shows how to draw a business card in the document foreground. To accomplish this task, do the following:
- Create a PDF document processor and add an empty document to it by calling the [CreateEmptyDocument](#) method.
 - Create PDF graphics represented by an instance of the DevExpress.Pdf.PdfGraphics class by calling the [CreateGraphics](#) method. To access DevExpress.Pdf.PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.
 - To draw an image on the PDF page, call the DevExpress.Pdf.PdfGraphics.DrawImage method for the specified image at the specified location with the specified size.
 - To draw text, call the DevExpress.Pdf.PdfGraphics.DrawString method at the specified location with the specified **Brush** and **Font** objects.
 - Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer.",
                                font3, black, 180, 300);
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF document with no pages using the file path.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub CreateEmptyDocument(  
    ByVal path As String  
)
```

C#

```
public void CreateEmptyDocument(  
    string path  
)
```

Parameters

path

A string that is the full path to the PDF document file.

Remarks

Use this method to create an empty document from scratch (instead of merging two existing documents). Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Additional Content](#) topic.

Note

When all operations with a document created using the overloaded **CreateEmptyDocument** method are completed, you need to close the document either by calling the [CloseDocument](#) method or disposing of the [PdfDocumentProcessor](#) instance.

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

To generate a document using the Document Creation API:

- Create an empty document with no pages by calling one of the [CreateEmptyDocument](#) overload methods (e.g., using a file path).
- Create PDF graphics represented by an instance of the DevExpress.Pdf.PdfGraphics class, calling the [CreateGraphics](#) method. To access DevExpress.Pdf.PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer.",
                                font3, black, 168, 300);
            }
        }
    }
}
```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF document with no pages using a stream.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub CreateEmptyDocument(
    ByVal stream As Stream
)
```

C#

```
public void CreateEmptyDocument(
    Stream stream
)
```

Parameters

stream

A [System.IO.Stream](#) class descendant specifying the stream to which the PDF empty document should be written.

Remarks

Use this method to create an empty document from scratch (instead of merging two existing documents). The empty document is created using a stream.

Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Additional Content](#) topic.

Note

When all operations with a document created using the overloaded **CreateEmptyDocument** method are completed, you need to close the document either by calling the [CloseDocument](#) method or disposing of the [PdfDocumentProcessor](#) instance.

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF document with no pages using the stream and PDF save options (containing encryption settings and a signature) that will be applied to the document when it is saved.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic

```
Public Sub CreateEmptyDocument(  
    ByVal stream As Stream,  
    ByVal saveOptions As PdfSaveOptions  
)
```

```
C#  
public void CreateEmptyDocument(  
    Stream stream,  
    PdfSaveOptions saveOptions  
)
```

Parameters
stream
A [System.IO.Stream](#) class descendant specifying the stream to which the empty PDF document should be written.
saveOptions
A DevExpress.Pdf.PdfSaveOptions object that contains settings to encrypt and sign an empty document.

Remarks
Use this method to create an empty document from scratch (instead of merging two existing documents). The document is created using the stream and PDF save options represented by the DevExpress.Pdf.PdfSaveOptions object. PDF save options will be applied to the document when it is saved. Using these options, you can access the encryption options (the DevExpress.Pdf.PdfSaveOptions.EncryptionOptions property) to protect an empty document with a password and permissions. You can also sign a document using the DevExpress.Pdf.PdfSaveOptions.Signature property.
Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Additional Content](#) topic.

Note

When all operations with a document created using the overloaded **CreateEmptyDocument** method are completed, you need to close the document either by calling the [CloseDocument](#) method or disposing of the [PdfDocumentProcessor](#) instance.

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at

<http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

To generate a document using the Document Creation API:

- Create an empty document with no pages by calling one of the [CreateEmptyDocument](#) overload methods (e.g., using a file path).
- Create PDF graphics represented by an instance of the `DevExpress.Pdf.PdfGraphics` class, calling the [CreateGraphics](#) method. To access `DevExpress.Pdf.PdfGraphics`, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer."
                                );
            }
        }
    }
}
```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF document with no pages using the file path and PDF save options (containing encryption settings and a signature) that will be applied to the document when it is saved.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub CreateEmptyDocument(
    ByVal path As String,
    ByVal saveOptions As PdfSaveOptions
)

```

C#

```

public void CreateEmptyDocument(
    string path,
    PdfSaveOptions saveOptions
)

```

Parameters

path

A string that is the full path to the PDF document file.

saveOptions

A DevExpress.Pdf.PdfSaveOptions object that contains settings to encrypt and sign an empty document.

Remarks

Use this method to create an empty document from scratch (instead of merging two existing documents). The document is created using the file path and PDF save options represented by the `DevExpress.Pdf.PdfSaveOptions` object. The PDF save options will be applied to the document when it is saved. Using the PDF save options, you can access the encryption options (the `DevExpress.Pdf.PdfSaveOptions.EncryptionOptions` property) to protect an empty document with a password and permissions. You can also sign a document using the `DevExpress.Pdf.PdfSaveOptions.Signature` property.

Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Additional Content](#) topic.

Note

When all operations with a document created using the overloaded **CreateEmptyDocument** method are completed, you need to close the document either by calling the [CloseDocument](#) method or disposing of the [PdfDocumentProcessor](#) instance.

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

To generate a document using the Document Creation API:

- Create an empty document with no pages by calling one of the [CreateEmptyDocument](#) overload methods (e.g., using a file path).
- Create PDF graphics represented by an instance of the `DevExpress.Pdf.PdfGraphics` class, calling the [CreateGraphics](#) method. To access `DevExpress.Pdf.PdfGraphics`, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer.")
            }
        }
    }
}
```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF with no pages using a stream and PDF creation options.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub CreateEmptyDocument(
    ByVal stream As Stream,
    ByVal creationOptions As PdfCreationOptions
)

```

C#

```

public void CreateEmptyDocument(
    Stream stream,
    PdfCreationOptions creationOptions
)

```

Parameters

stream

A [System.IO.Stream](#) class descendant specifying the stream to which the PDF empty document should be written.

creationOptions

A DevExpress.Pdf.PdfCreationOptions object that contains PDF compatibility and font embedding options.

Remarks

Use this method to create an empty document from scratch. The empty document is created using a stream and PDF creation settings represented by the `DevExpress.Pdf.PdfCreationOptions` object. Using the PDF creation settings, you can create an empty PDF document by setting the `DevExpress.Pdf.PdfCreationOptions.Compatibility` property to one of the `DevExpress.Pdf.PdfCompatibility` enumeration values. You can also disable embedding all fonts when the document is created using the `DevExpress.Pdf.PdfCreationOptions.DisableEmbeddingAllFonts` property or disable embedding certain fonts using the `DevExpress.Pdf.PdfCreationOptions.NotEmbeddedFontFamilies` property.

Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Additional Content](#) topic.

Note

When all operations with a document created using the overloaded **CreateEmptyDocument** method are completed, you need to close the document either by calling the [CloseDocument](#) method or disposing of the [PdfDocumentProcessor](#) instance.

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

- To generate a document using the Document Creation API:
- Create an empty document with no pages by calling one of the [CreateEmptyDocument](#) overload methods (e.g., using a file path).
 - Create PDF graphics represented by an instance of the `DevExpress.Pdf.PdfGraphics` class, calling the [CreateGraphics](#) method. To access `DevExpress.Pdf.PdfGraphics`, you need to reference the **DevExpress.Pdf.Drawing** assembly.
 - Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
 - Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer."
                                );
            }
        }
    }
}
```

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF document with no pages using the file path and PDF creation options.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub CreateEmptyDocument(
    ByVal path As String,
    ByVal creationOptions As PdfCreationOptions
)

```

C#

```

public void CreateEmptyDocument(
    string path,
    PdfCreationOptions creationOptions
)

```

Parameters

path
A string that is the full path to the PDF document file.

creationOptions
A DevExpress.Pdf.PdfCreationOptions object that contains PDF compatibility and font embedding options.

Remarks

Use this method to create an empty document from scratch. The empty document is created using the file path and PDF creation settings represented by the `DevExpress.Pdf.PdfCreationOptions` object. Using the PDF creation settings, you can create a PDF document by setting the `DevExpress.Pdf.PdfCreationOptions.Compatibility` property to one of the `DevExpress.Pdf.PdfCompatibility` enumeration values. You can also disable embedding all fonts when the document is created using the `DevExpress.Pdf.PdfCreationOptions.DisableEmbeddingAllFonts` property or disable embedding certain fonts using the `DevExpress.Pdf.PdfCreationOptions.NotEmbeddedFontFamilies` property.

Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Additional Content](#) topic.

Note

When all operations with a document created using the overloaded **CreateEmptyDocument** method are completed, you need to close the document either by calling the [CloseDocument](#) method or disposing of the [PdfDocumentProcessor](#) instance.

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

To generate a document using the Document Creation API:

- Create an empty document with no pages by calling one of the [CreateEmptyDocument](#) overload methods (e.g., using a file path).
- Create PDF graphics represented by an instance of the `DevExpress.Pdf.PdfGraphics` class, calling the [CreateGraphics](#) method. To access `DevExpress.Pdf.PdfGraphics`, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer."
                                );
            }
        }
    }
}
```

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF document with no pages using a stream, PDF save options (containing encryption settings and a signature) that will be applied to the document when it is saved, and PDF creation options.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub CreateEmptyDocument(
    ByVal stream As Stream,
    ByVal saveOptions As PdfSaveOptions,
    ByVal creationOptions As PdfCreationOptions
)

```

C#

```

public void CreateEmptyDocument(
    Stream stream,
    PdfSaveOptions saveOptions,
    PdfCreationOptions creationOptions
)

```

Parameters

stream

A [System.IO.Stream](#) class descendant specifying the stream to which the PDF empty document should be written.

saveOptions

A DevExpress.Pdf.PdfSaveOptions object that contains settings to encrypt and sign an empty document.
creationOptions
A DevExpress.Pdf.PdfCreationOptions object that contains PDF compatibility and font embedding options.

Remarks

Use this method to create an empty document from scratch (instead of merging two existing documents). The document is created using a stream, PDF save options (represented by the DevExpress.Pdf.PdfSaveOptions object), and PDF creation options (represented by the DevExpress.Pdf.PdfCreationOptions object). The PDF save options will be applied to the document when it is saved.

The PDF save options allow you to access the encryption options to protect an empty document with a password and permissions (the DevExpress.Pdf.PdfSaveOptions.EncryptionOptions property). You can also sign a document using the DevExpress.Pdf.PdfSaveOptions.Signature property.

Using the PDF creation options, you can create an empty PDF document by setting the DevExpress.Pdf.PdfCreationOptions.Compatibility property to one of the DevExpress.Pdf.PdfCompatibility enumeration values. You can also disable embedding all fonts when the document is created using the DevExpress.Pdf.PdfCreationOptions.DisableEmbeddingAllFonts property or disable embedding certain fonts using the DevExpress.Pdf.PdfCreationOptions.NotEmbeddedFontFamilies property.

Note

When all operations with a document created using the overloaded **CreateEmptyDocument** method are completed, you need to close the document either by calling the [CloseDocument](#) method or disposing of the [PdfDocumentProcessor](#) instance.

Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Additional Content](#) topic.

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

To generate a document using the Document Creation API:

- Create an empty document with no pages by calling one of the [CreateEmptyDocument](#) overload methods (e.g., using a file path).
- Create PDF graphics represented by an instance of the DevExpress.Pdf.PdfGraphics class, calling the [CreateGraphics](#) method. To access DevExpress.Pdf.PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer."
                                );
            }
        }
    }
}
```

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

Creates an empty PDF document with no pages using the file path, PDF save options (containing encryption settings and a signature) that will be applied to the document when it is saved, and PDF creation options.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub CreateEmptyDocument(
    ByVal path As String,
    ByVal saveOptions As PdfSaveOptions,
    ByVal creationOptions As PdfCreationOptions
)

```

C#

```

public void CreateEmptyDocument(
    string path,
    PdfSaveOptions saveOptions,
    PdfCreationOptions creationOptions
)

```

Parameters

path

A string that is the full path to the PDF document file.

saveOptions

A DevExpress.Pdf.PdfSaveOptions object that contains settings to encrypt and sign an empty document.
creationOptions
A DevExpress.Pdf.PdfCreationOptions object that contains PDF compatibility and font embedding options.

Remarks

Use this method to create an empty document from scratch. The empty document is created using the file path, PDF save options (represented by the DevExpress.Pdf.PdfSaveOptions object) that will be applied to the document when it is saved, and PDF creation options (represented by the DevExpress.Pdf.PdfCreationOptions object).

The PDF save options allows you to access the encryption options to protect (the DevExpress.Pdf.PdfSaveOptions.EncryptionOptions property) a document with a password and permissions. You can also sign a document using the DevExpress.Pdf.PdfSaveOptions.Signature property.

Using the PDF creation options, you can create an empty PDF document by setting the DevExpress.Pdf.PdfCreationOptions.Compatibility property to one of the DevExpress.Pdf.PdfCompatibility enumeration values. You can also disable embedding all fonts when the document is created using the DevExpress.Pdf.PdfCreationOptions.DisableEmbeddingAllFonts property or disable embedding certain fonts using the DevExpress.Pdf.PdfCreationOptions.NotEmbeddedFontFamilies property.

Note

When all operations with a document created using the overloaded **CreateEmptyDocument** method are completed, you need to close the document either by calling the [CloseDocument](#) method or disposing of the [PdfDocumentProcessor](#) instance.

Then, you can continue to generate the document layout (e.g., append pages with graphics to the PDF document, generate bookmarks, and attach files) using the PDF document creation API. For more information, see the [Additional Content](#) topic.

The PDF specification does not describe empty documents. For this reason, most third-party PDF viewers cannot open such files. This does not apply to the DevExpress WinForms PDF Viewer and WPF PDF Viewer, which are less demanding concerning the validity of opened documents, and are capable of opening documents containing no pages.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

This example shows how to programmatically create a document with graphics using the PDF Document Creation API.

To generate a document using the Document Creation API:

- Create an empty document with no pages by calling one of the [CreateEmptyDocument](#) overload methods (e.g., using a file path).
- Create PDF graphics represented by an instance of the DevExpress.Pdf.PdfGraphics class, calling the [CreateGraphics](#) method. To access DevExpress.Pdf.PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- Draw the graphic content (e.g., an image, a string, lines, polygons) by calling the corresponding **Draw** method.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer."
                                );
            }
        }
    }
}
```

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides")
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateEmptyDocument Overload List](#)

CreateGraphics Method

Creates a new instance of the DevExpress.Pdf.PdfGraphics used as a drawing context that can draw graphics on a PDF document.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CreateGraphics() As PdfGraphics
```

C#

```
public PdfGraphics CreateGraphics()
```

Return Value

A DevExpress.Pdf.PdfGraphics object used as a drawing context that can be used to draw on the document.

Remarks

Use this method to create an instance of the DevExpress.Pdf.PdfGraphics. This class declares the methods used to create an image (DevExpress.Pdf.PdfGraphics.DrawImage), text (DevExpress.Pdf.PdfGraphics.DrawString) and graphic elements on a document.

To add graphic content to a new PDF page, call one of the [RenderNewPage](#) overload methods with the specified paper size and created graphics.

To add graphics to an existing document, call the DevExpress.Pdf.PdfGraphics.AddToPageForeground and DevExpress.Pdf.PdfGraphics.AddToPageBackground methods.

For more details, see the [PDF Graphics](#) section.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

The PDF Document Processor allows you to draw graphic primitives on the PDF page, such as lines, Bezier curves, polygons, ellipses, paths, an image and a string.

- This example shows how to draw a business card in the document foreground. To accomplish this task, do the following:
- Create a PDF document processor and add an empty document to it by calling the [CreateEmptyDocument](#) method.
 - Create PDF graphics represented by an instance of the DevExpress.Pdf.PdfGraphics class by calling the [CreateGraphics](#) method. To access DevExpress.Pdf.PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.
 - To draw an image on the PDF page, call the DevExpress.Pdf.PdfGraphics.DrawImage method for the specified image at the specified location with the specified size.
 - To draw text, call the DevExpress.Pdf.PdfGraphics.DrawString method at the specified location with the specified **Brush** and **Font** objects.
 - Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer.")
            }
        }
    }
}
```

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

CreateTiff Method

Exports a PDF document to a TIFF image using a stream, and the image's largest edge length.

Overload List

Name	Description
void CreateTiff(Stream stream, int largestEdgeLength)	Exports a PDF document to a TIFF image using a stream, and the image's largest edge length.
void CreateTiff(string fileName, int largestEdgeLength)	Exports a PDF document to a TIFF image using the image's file path, and the image's largest edge length.
void CreateTiff(Stream stream, int largestEdgeLength, IEnumerable<Int32> pageNumbers)	Exports a PDF document to a TIFF image using a stream, the image's largest edge length, and page numbers.
void CreateTiff(string fileName, int largestEdgeLength, IEnumerable<Int32> pageNumbers)	Exports a PDF document to a TIFF image using the image's file path, image's largest edge length, and page numbers.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CreateTiff Overload List](#)

Exports a PDF document to a TIFF image using a stream, and the image's largest edge length.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub CreateTiff(  
    ByVal stream As Stream,  
    ByVal largestEdgeLength As Integer  
)
```

C#

```
public void CreateTiff(  
    Stream stream,  
    int largestEdgeLength  
)
```

Parameters

stream
A [System.IO.Stream](#) object that is a stream to which a TIFF image should be written.
largestEdgeLength
An integer value, specifying the length of the image's largest dimension, in pixels.

Remarks

The **largestEdgeLength** parameter determines the output image height for pages in the portrait orientation and width - for landscape pages.

 **Azure Compatibility**

The **CreateTiff** method works on Azure Roles (Worker and/or Web) and does not work on Azure web site.

Example

The following example shows how to convert pages to a multi-page Tiff image.

 **Show Me**

The complete sample project is available at <https://github.com/DevExpress-Examples/how-to-export-a-pdf-document-to-multi-page-tiff-and-bitmap>

C#

```
using DevExpress.Pdf;
namespace ExportToTiff {
    class Program {
        static void Main(string[] args) {
            int largestEdgeLength = 1000;
            int[] pageNumbers = new int[] { 1, 3, 5 };
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("..\\..\\Document.pdf");
                // Export pages to a multi-page tiff image.
                processor.CreateTiff("..\\..\\Image.tiff", largestEdgeLength, pageNumbers);
            }
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
Namespace ExportToTiff
    Class Program
        Private Shared Sub Main(ByVal args As String())
            Dim largestEdgeLength As Integer = 1000
            Dim pageNumbers As Integer() = New Integer() {1, 3, 5}
            Using processor As PdfDocumentProcessor = New PdfDocumentProcessor()
                processor.LoadDocument("..\\..\\Document.pdf")
                processor.CreateTiff("..\\..\\Image.tiff", largestEdgeLength, pageNumbers)
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CreateTiff Overload List](#)

Exports a PDF document to a TIFF image using the image's file path, and the image's largest edge length.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub CreateTiff(
    ByVal fileName As String,
    ByVal largestEdgeLength As Integer
)
```

C#

```
public void CreateTiff(
    string fileName,
    int largestEdgeLength
)
```

Parameters

fileName

A [System.String](#) which specifies the file name (including the full path) for the created Tiff image.

largestEdgeLength

An integer value, specifying the length of the image's largest dimension, in pixels.

Remarks

The **largestEdgeLength** parameter determines the output image height for pages in the portrait orientation and width - for landscape pages.

Azure Compatibility

The **CreateTiff** method works on Azure Roles (Worker and/or Web) and does not work on Azure web site.

Example

The following example shows how to convert pages to a multi-page Tiff image.

Show Me

The complete sample project is available at <https://github.com/DevExpress-Examples/how-to-export-a-pdf-document-to-multi-page-tiff-and-bitmap>

C#

```
using DevExpress.Pdf;
namespace ExportToTiff {
    class Program {
        static void Main(string[] args) {
            int largestEdgeLength = 1000;
            int[] pageNumbers = new int[] { 1, 3, 5 };
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../..\\Document.pdf");
                // Export pages to a multi-page tiff image.
                processor.CreateTiff("../..\\Image.tiff", largestEdgeLength, pageNumbers);
            }
        }
    }
}
```

Visual Basic

```
Imports DevExpress.Pdf
Namespace ExportToTiff
    Class Program
        Private Shared Sub Main(ByVal args As String())
            Dim largestEdgeLength As Integer = 1000
            Dim pageNumbers As Integer() = New Integer() {1, 3, 5}
            Using processor As PdfDocumentProcessor = New PdfDocumentProcessor()
                processor.LoadDocument("../..\\Document.pdf")
                processor.CreateTiff("../..\\Image.tiff", largestEdgeLength, pageNumbers)
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.CreateTiff Overload List](#)

Exports a PDF document to a TIFF image using a stream, the image's largest edge length, and page numbers.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub CreateTiff(
    ByVal stream As Stream,
    ByVal largestEdgeLength As Integer,
    ByVal pageNumbers As IEnumerable(of Int32)
)
```

C#

```
public void CreateTiff(  
    Stream stream,  
    int largestEdgeLength,  
    IEnumerable<Int32> pageNumbers  
)
```

Parameters

stream
A [System.IO.Stream](#) object that is a stream to which the TIFF image should be written.

largestEdgeLength
An integer value, specifying the length of the image's largest dimension, in pixels.

pageNumbers
A list of page numbers which implement the `System.Collections.Generic.IEnumerable<System.Int32>` interface.

Remarks

The **largestEdgeLength** parameter determines the output image height for pages in the portrait orientation and width - for landscape pages.

Azure Compatibility

The **CreateTiff** method works on Azure Roles (Worker and/or Web) and does not work on Azure web site.

Example

The following example shows how to convert pages to a multi-page Tiff image.

Show Me

The complete sample project is available at <https://github.com/DevExpress-Examples/how-to-export-a-pdf-document-to-multi-page-tiff-and-bitmap>

C#

```
using DevExpress.Pdf;  
namespace ExportToTiff {  
    class Program {  
        static void Main(string[] args) {  
            int largestEdgeLength = 1000;  
            int[] pageNumbers = new int[] { 1, 3, 5 };  
            // Create a PDF Document Processor.  
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {  
                // Load a document.  
                processor.LoadDocument("..\\..\\Document.pdf");  
                // Export pages to a multi-page tiff image.  
                processor.CreateTiff("..\\..\\Image.tiff", largestEdgeLength, pageNumbers);  
            }  
        }  
    }  
}
```

Visual Basic

```
Imports DevExpress.Pdf  
Namespace ExportToTiff  
    Class Program  
        Private Shared Sub Main(ByVal args As String())  
            Dim largestEdgeLength As Integer = 1000  
            Dim pageNumbers As Integer() = New Integer() {1, 3, 5}  
            Using processor As PdfDocumentProcessor = New PdfDocumentProcessor()  
                processor.LoadDocument("..\\..\\Document.pdf")  
                processor.CreateTiff("..\\..\\Image.tiff", largestEdgeLength, pageNumbers)  
            End Using  
        End Sub  
    End Class  
End Namespace
```

See Also
[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.CreateTiff Overload List](#)

Exports a PDF document to a TIFF image using the image's file path, image's largest edge length, and page numbers.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub CreateTiff(  
    ByVal fileName As String,  
    ByVal largestEdgeLength As Integer,  
    ByVal pageNumbers As IEnumerable(of Int32)  
)
```

C#


```
public void CreateTiff(  
    string fileName,  
    int largestEdgeLength,  
    IEnumerable<Int32> pageNumbers  
)
```

Parameters

- fileName*
A [System.String](#) which specifies the file name (including the full path) for the created Tiff image.
- largestEdgeLength*
An integer value, specifying the length of the image's largest dimension, in pixels.
- pageNumbers*
A list of page numbers which implement the System.Collections.Generic.IEnumerable<System.Int32> interface.

Remarks

The **largestEdgeLength** parameter determines the output image height for pages in the portrait orientation and width - for landscape pages.

 Azure Compatibility	
The CreateTiff method works on Azure Roles (Worker and/or Web) and does not work on Azure web site.	

Example

The following example shows how to convert pages to a multi-page Tiff image.

 Show Me	
The complete sample project is available at https://github.com/DevExpress-Examples/how-to-export-a-pdf-document-to-multi-page-tiff-and-bitmap	

C#

```
using DevExpress.Pdf;  
namespace ExportToTiff {  
    class Program {  
        static void Main(string[] args) {  
            int largestEdgeLength = 1000;  
            int[] pageNumbers = new int[] { 1, 3, 5 };  
            // Create a PDF Document Processor.  
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {  
                // Load a document.  
                processor.LoadDocument("..\\..\\Document.pdf");  
                // Export pages to a multi-page tiff image.  
                processor.CreateTiff("..\\..\\Image.tiff", largestEdgeLength, pageNumbers);  
            }  
        }  
    }  
}
```

Visual Basic

```
Imports DevExpress.Pdf
Namespace ExportToTiff
    Class Program
        Private Shared Sub Main(ByVal args As String())
            Dim largestEdgeLength As Integer = 1000
            Dim pageNumbers As Integer() = New Integer() {1, 3, 5}
            Using processor As PdfDocumentProcessor = New PdfDocumentProcessor()
                processor.LoadDocument("../..\Document.pdf")
                processor.CreateTiff("../..\Image.tiff", largestEdgeLength, pageNumbers)
            End Using
        End Sub
    End Class
End Namespace
```

See Also[PdfDocumentProcessor Class](#)[PdfDocumentProcessor Members](#)[DevExpress.Pdf Namespace](#)[PdfDocumentProcessor.CreateTiff Overload List](#)

DeleteAttachment Method

Returns a value indicating if the attachment is deleted from a PDF document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function DeleteAttachment(  
    ByVal attachment As PdfFileAttachment  
) As Boolean
```

C#

```
public bool DeleteAttachment(  
    PdfFileAttachment attachment  
)
```

Parameters

attachment

A DevExpress.Pdf.PdfFileAttachment object which contains attachment settings.

Return Value

If the attachment is deleted from a PDF document, the value is **true**; otherwise, **false**.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

DeleteMarkupAnnotation Method

Deletes a markup annotation from a page.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub DeleteMarkupAnnotation(  
    ByVal annotationData As PdfMarkupAnnotationData  
)
```

C#

```
public void DeleteMarkupAnnotation(  
    PdfMarkupAnnotationData annotationData  
)
```

Parameters

annotationData

A DevExpress.Pdf.PdfMarkupAnnotationData object that is the text markup annotation that will be deleted.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

DeleteMarkupAnnotations Method

Deletes markup annotations specified in the DevExpress.Pdf.PdfMarkupAnnotationData collection.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub DeleteMarkupAnnotations(  
    ByVal annotations As IEnumerable(of PdfMarkupAnnotationData)  
)
```

C#

```
public void DeleteMarkupAnnotations(  
    IEnumerable<PdfMarkupAnnotationData> annotations  
)
```

Parameters

annotations
A collection of DevExpress.Pdf.PdfMarkupAnnotationData objects that represent text markup annotations that will be deleted.

Remarks

To delete a specific markup annotation, call the [DeleteMarkupAnnotation](#) method.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T545688>.

This example shows how to delete all markup annotation from document pages.

To retrieve all text markup annotations in a page, call the [GetMarkupAnnotationData](#) method with a specified page number.

To delete all markup annotations, call the [DeleteMarkupAnnotations](#) method passing the markup annotation list as an argument to this method.

C#

```
(Program.cs)  
using DevExpress.Pdf;  
using System.Collections.Generic;  
namespace RemoveAllMarkupAnnotations {  
    class Program {  
        static void Main(string[] args) {  
            // Create a PDF Document Processor.  
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {  
                // Load a document.  
                processor.LoadDocument("../\\..\\Document.pdf");  
                // Get a list of annotations in the document pages.  
                for (int i = 0; i <= processor.Document.Pages.Count; i++) {  
                    IList<PdfMarkupAnnotationData> annotations = processor.GetMarkupAnnotationData(i);  
                    // Delete all text markup annotations from document pages.  
                    processor.DeleteMarkupAnnotations(annotations);  
                    // Save the result document.  
                    processor.SaveDocument("../\\..\\Result.pdf");  
                }  
            }  
        }  
    }  
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System.Collections.Generic
Namespace RemoveAllMarkupAnnotations
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../..\Document.pdf")
                ' Get a list of annotations in the document pages.
                For i As Integer = 0 To processor.Document.Pages.Count
                    Dim annotations As IList(Of PdfMarkupAnnotationData) = processor.GetMarkupAnnotationData(i)
                    ' Delete all text markup annotations from document pages.
                    processor.DeleteMarkupAnnotations(annotations)
                    ' Save the result document.
                    processor.SaveDocument("../..\Result.pdf")
                Next i
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

DeletePage Method

Deletes the specified page from the current document.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub DeletePage(  
    ByVal pageNumber As Integer  
)
```

C#

```
public void DeletePage(  
    int pageNumber  
)
```

Parameters

pageNumber
An integer value, specifying the page number.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T114310>.

This example illustrates how to use the PDF Document API component for deleting pages from a document.

In this example, the [DeletePage](#) method is called 55 times to delete each odd numbered page in the **TextDelete** document (contains 109 pages), starting from the last odd numbered page.

The result is saved to the **Deleted** document, which contains only even pages, by calling the [SaveDocument](#) method.

C#

```
(Program.cs)  
using DevExpress.Pdf;  
// ...  
class Program {  
    static void Main(string[] args) {  
        using (PdfDocumentProcessor pdfDocumentProcessor = new PdfDocumentProcessor()) {  
            pdfDocumentProcessor.LoadDocument("..\\..\\docs\\TextDelete.pdf");  
            for (int i = pdfDocumentProcessor.Document.Pages.Count / 2; i >= 0; i--)  
                pdfDocumentProcessor.DeletePage(i * 2 + 1);  
            pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Deleted.pdf");  
        }  
    }  
}
```

Visual Basic

```
(Program.vb)  
Imports DevExpress.Pdf  
' ...  
Friend Class Program  
    Shared Sub Main(ByVal args() As String)  
        Using pdfDocumentProcessor As New PdfDocumentProcessor()  
            pdfDocumentProcessor.LoadDocument("..\\..\\docs\\TextDelete.pdf")  
            For i As Integer = pdfDocumentProcessor.Document.Pages.Count \ 2 To 0 Step -1  
                pdfDocumentProcessor.DeletePage(i * 2 + 1)  
            Next i  
            pdfDocumentProcessor.SaveDocument("..\\..\\docs\\Deleted.pdf")  
        End Using  
    End Sub  
End Class
```

See Also[PdfDocumentProcessor Class](#)[PdfDocumentProcessor Members](#)[DevExpress.Pdf Namespace](#)

Export Method

Exports interactive form data to the file using the specified form data format.

Overload List

Name	Description
void Export(string fileName, PdfFormDateFormat format)	Exports interactive form data to the file using the specified form data format.
void Export(Stream stream, PdfFormDateFormat format)	Exports interactive form data to the specified stream using specified form data format.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.Export Overload List](#)

Exports interactive form data to the file using the specified form data format.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Export(  
    ByVal fileName As String,  
    ByVal format As PdfFormDateFormat  
)
```

C#

```
public void Export(  
    string fileName,  
    PdfFormDateFormat format  
)
```

Parameters

fileName
A [System.String](#) specifying the path to the file to which interactive form data should be exported.
format
A DevExpress.Pdf.PdfFormDateFormat enumeration value that represents one of the supported formats for form data values.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.Export Overload List](#)

Exports interactive form data to the specified stream using specified form data format.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Export(  
    ByVal stream As Stream,  
    ByVal format As PdfFormDateFormat  
)
```

C#

```
public void Export(  
    Stream stream,  
    PdfFormDateFormat format  
)
```

Parameters

stream

A [System.IO.Stream](#) class descendant specifying the stream containing the document to which interactive form data should be exported.

format

A DevExpress.Pdf.PdfFormDateFormat enumeration value that represents one of the supported formats for form data values.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.Export Overload List](#)

FindText Method

Searches for the specified text in the current document with default parameters.

Overload List

Name	Description
PdfTextSearchResults FindText(string text)	Searches for the specified text in the current document with default parameters.
PdfTextSearchResults FindText(string text, PdfTextSearchParameters parameters)	Searches for the specified text in the current document with the applied parameters.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.FindText Overload List](#)

Searches for the specified text in the current document with default parameters.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function FindText(  
    ByVal text As String  
) As PdfTextSearchResults
```

C#

```
public PdfTextSearchResults FindText(  
    string text  
)
```

Parameters

text
A [System.String](#) value, specifying the text to find in the PDF.

Return Value

A DevExpress.Pdf.PdfTextSearchResults object.

Remarks

The overloaded **FindText** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.FindText Overload List](#)

Searches for the specified text in the current document with the applied parameters.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function FindText(  
    ByVal text As String,  
    ByVal parameters As PdfTextSearchParameters  
) As PdfTextSearchResults
```

C#

```
public PdfTextSearchResults FindText(  
    string text,  
    PdfTextSearchParameters parameters  
)
```

Parameters

text

A [System.String](#) value, specifying the text to find in the PDF.

parameters

A DevExpress.Pdf.PdfTextSearchParameters object.

Return Value

A DevExpress.Pdf.PdfTextSearchResults object.

Remarks

The overloaded **FindText** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.FindText Overload List](#)

FlattenForm Method

Flattens an entire interactive form.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Function FlattenForm() As [Boolean](#)

C#

public [bool](#) FlattenForm()

Return Value

true If the interactive form is flattened successfully; **false** If the document doesn't contain an interactive form that should be flattened.

Remarks

You can call the **FlattenForm** method the following way:

C#

```
using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {  
    processor.LoadDocument("Demo.pdf");  
    if (processor.FlattenForm())  
        processor.SaveDocument("Result.pdf");  
}
```

Visual Basic

```
Using processor As New PdfDocumentProcessor()  
    processor.LoadDocument("Demo.pdf")  
    If processor.FlattenForm() Then  
        processor.SaveDocument("Result.pdf")  
    End If  
End Using
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

FlattenFormField Method

Flattens a specific form field on an interactive form by its name.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function FlattenFormField(  
    ByVal name As String  
) As Boolean
```

C#

```
public bool FlattenFormField(  
    string name  
)
```

Parameters

name
A string that is the name of the form field to be flattened.

Return Value

true, if the interactive form field is flattened successfully; **false** If the document doesn't contain a form field that should be flattened.

Remarks

To obtain a list of interactive field names in a document, call the [GetFormFieldNames](#) method which returns a string collection.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T446039>.

This example shows how to flatten interactive form fields (e.g., text fields, list boxes).

To flatten a whole interactive form, call the [FlattenForm](#) method. If the interactive form is flattened successfully, this method returns **true**. If the document doesn't contain an interactive form that should be flattened, this method returns **false**.

To flatten a particular form field by its name, call the [FlattenFormField](#) method with a field name used as a parameter. This method tries to find the interactive form field in a document using the field name. If this form field is found in the document, then it will be flattened and this method returns **true**; otherwise, it returns **false**.

C#

```

(Program.cs)
using DevExpress.Pdf;
using System;
namespace FlattenInteractiveForm {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document with an interactive form.
                processor.LoadDocument("../..\..\Document.pdf");
                // Flatten a form field by its name
                if (processor.FlattenFormField("Nationality"))
                    // Save a document with the flattened form field.
                    processor.SaveDocument("../..\..\Result1.pdf");
                // Show a message if the form field was not found in a document.
                else
                    Console.WriteLine("A document does not contain a form field with the specified name to be flattened.");
                // Flatten a whole interactive form.
                if (processor.FlattenForm())
                    // Save a document with the flattened form.
                    processor.SaveDocument("../..\..\Result2.pdf");
                // Show a message if the interactive was not found in a document.
                else
                    Console.WriteLine("A document does not contain an interactive form to be flattened.");
            }
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Namespace FlattenInteractiveForm
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document with an interactive form.
                processor.LoadDocument("../..\..\Document.pdf")
                ' Flatten a form field by its name
                If processor.FlattenFormField("Nationality") Then
                    ' Save a document with the flattened form field.
                    processor.SaveDocument("../..\..\Result1.pdf")
                    ' Show a message if the form field was not found in a document.
                Else
                    Console.WriteLine("A document does not contain a form field with the specified name to be flattened.")
                End If
                ' Flatten a whole interactive form.
                If processor.FlattenForm() Then
                    ' Save a document with the flattened form.
                    processor.SaveDocument("../..\..\Result2.pdf")
                    ' Show a message if the interactive was not found in a document.
                Else
                    Console.WriteLine("A document does not contain an interactive form to be flattened.")
                End If
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)

GetFormData Method

Returns an object containing values of interactive form data fields.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetFormData() As PdfFormData
```

C#

```
public PdfFormData GetFormData()
```

Return Value

A DevExpress.Pdf.PdfFormData object.

Remarks

After the interactive form is changed (e.g., using one of the following operations: adding new interactive form fields, interactive form flattening, merging documents with interactive forms, or deleting pages with interactive form fields), the previously obtained interactive form data becomes invalid. To get valid interactive form data from a document, call the **GetFormData** method again.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the DevExpress.Pdf.PdfFormData.Value property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#

```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic

```
(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

GetFormFieldNames Method

Returns a list of interactive field names in a document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetFormFieldNames() As IList(of String)
```

C#

```
public IList<String> GetFormFieldNames()
```

Return Value

A collection of [System.String](#) objects that is a list of interactive field names in a document.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

GetImages Method

Retrieves the images found within the specified document area.

Overload List

Name	Description
IList<Bitmap> GetImages(PdfDocumentArea area)	Retrieves the images found within the specified document area.
IList<Bitmap> GetImages(PdfDocumentPosition startPosition, PdfDocumentPosition endPosition)	Retrieves the images found within the specified document positions.
IList<Bitmap> GetImages(PdfDocumentArea area, Single imageResolution)	Retrieves images found within the specified document area using image resolution.
IList<Bitmap> GetImages(PdfDocumentPosition startPosition, PdfDocumentPosition endPosition, Single imageResolution)	Retrieves the images found within the specified document positions using image resolution.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.GetImages Overload List](#)

Retrieves the images found within the specified document area.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetImages(  
    ByVal area As PdfDocumentArea  
) As IList(of Bitmap)
```

C#

```
public IList<Bitmap> GetImages(  
    PdfDocumentArea area  
)
```

Parameters


area
A DevExpress.Pdf.PdfDocumentArea object.

Return Value

A collection of [System.Drawing.Bitmap](#) objects.


Remarks

The overloaded **GetImages** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

 **Important**

The **GetImages** method does not work in **Partial Trust** environments and requires the **Full Trust** permission level.

Example

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T161599>.

This example illustrates the use of the [GetImages](#) method for obtaining document bitmaps in code by using [PDF Document API](#).

C#

```
(Program.cs)
using System;
using System.Collections.Generic;
using System.Drawing;
using DevExpress.Pdf;
// ...

static void Main(string[] args) {
    PdfDocumentProcessor processor = new PdfDocumentProcessor();
    processor.LoadDocument(@"..\..\..\Demo.pdf");
    int xCount = 8;
    int yCount = 2;
    double cardWidth = 150.5; // Measured in points (equals 2.09 inches).
    double cardHeight = 442; // Measured in points (equals 6.138 inches).
    double xMargin = 122; // Measured in points (equals 1.694 inches).
    double yMargin = 77; // Measured in points (equals 1.069 inches).
    double yCoord = yMargin;
    for (int y = 0; y < yCount; y++, yCoord += cardHeight) {
        double xCoord = xMargin;
        for (int x = 0; x < xCount; x++, xCoord += cardWidth) {
            PdfDocumentArea area = new PdfDocumentArea(1,
                new PdfRectangle(xCoord, yCoord, xCoord + cardWidth, yCoord + cardHeight));
            IList<Bitmap> bitmaps = processor.GetImages(area);
            if (bitmaps.Count != 0) {
                bitmaps[0].Save(String.Format(@"{0}_{1}.bmp", x, y));
                bitmaps[0].Dispose();
            }
            Console.WriteLine(bitmaps.Count.ToString());
        }
    }
}
```

Visual Basic

```

(Program.vb)
Imports System
Imports System.Collections.Generic
Imports System.Drawing
Imports DevExpress.Pdf
' ...

Shared Sub Main(ByVal args() As String)
    Dim processor As New PdfDocumentProcessor()
    processor.LoadDocument("../..\\Demo.pdf")
    Dim xCount As Integer = 8
    Dim yCount As Integer = 2
    Dim cardWidth As Double = 150.5 ' Measured in points (equals 2.09 inches).
    Dim cardHeight As Double = 442 ' Measured in points (equals 6.138 inches).
    Dim xMargin As Double = 122 ' Measured in points (equals 1.694 inches).
    Dim yMargin As Double = 77 ' Measured in points (equals 1.069 inches).
    Dim yCoord As Double = yMargin
    Dim y As Integer = 0
    Do While y < yCount
        Dim xCoord As Double = xMargin
        Dim x As Integer = 0
        Do While x < xCount
            Dim area As New PdfDocumentArea(1, New PdfRectangle(xCoord, yCoord, xCoord + cardWidth, yCoord + cardHeight))
            Dim bitmaps As IList(Of Bitmap) = processor.GetImages(area)
            If bitmaps.Count <> 0 Then
                bitmaps(0).Save(String.Format("{0}_{1}.bmp", x, y))
                bitmaps(0).Dispose()
            End If
            Console.WriteLine(bitmaps.Count.ToString())
            x += 1
            xCoord += cardWidth
        Loop
        y += 1
        yCoord += cardHeight
    Loop
End Sub

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.GetImages Overload List](#)

Retrieves the images found within the specified document positions.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Function GetImages(
    ByVal startPosition As PdfDocumentPosition,
    ByVal endPosition As PdfDocumentPosition
) As IList(of Bitmap)

```

C#

```

public IList<Bitmap> GetImages(
    PdfDocumentPosition startPosition,
    PdfDocumentPosition endPosition
)

```

Parameters

startPosition

A DevExpress.Pdf.PdfDocumentPosition object.

endPosition

A DevExpress.Pdf.PdfDocumentPosition object.

Return Value

A collection of [System.Drawing.Bitmap](#) objects.

Remarks

The overloaded **GetImages** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

Important

The **GetImages** method does not work in **Partial Trust** environments and requires the **Full Trust** permission level.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.GetImages Overload List](#)

[How to: Extract Images from a Document](#)

Retrieves images found within the specified document area using image resolution.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetImages(  
    ByVal area As PdfDocumentArea,  
    ByVal imageResolution As Single  
) As IList(of Bitmap)
```

C#

```
public IList<Bitmap> GetImages(  
    PdfDocumentArea area,  
    Single imageResolution  
)
```

Parameters

area

A DevExpress.Pdf.PdfDocumentArea object that represents a document area.

imageResolution

A [System.Single](#) object that represents the value, in dots per inch, for the image resolution.

Return Value

A collection of [System.Drawing.Bitmap](#) objects.

Remarks

The overloaded **GetImages** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

Important

The **GetImages** method does not work in **Partial Trust** environments and requires the **Full Trust** permission level.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.GetImages Overload List](#)

Retrieves the images found within the specified document positions using image resolution.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetImages(  
    ByVal startPosition As PdfDocumentPosition,  
    ByVal endPosition As PdfDocumentPosition,  
    ByVal imageResolution As Single  
) As IList(of Bitmap)
```

C#

```
public IList<Bitmap> GetImages(  
    PdfDocumentPosition startPosition,  
    PdfDocumentPosition endPosition,  
    Single imageResolution  
)
```

Parameters

startPosition

A DevExpress.Pdf.PdfDocumentPosition object that is the initial document position, starting from which, the images is obtained.

endPosition

A DevExpress.Pdf.PdfDocumentPosition object that is the final document position, starting from which, images are not obtained.

imageResolution

A [System.Single](#) object that represents the value, in dots per inch, for the image resolution.

Return Value

A collection of [System.Drawing.Bitmap](#) objects.

Remarks

The overloaded **GetImages** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

 **Important**

The **GetImages** method does not work in **Partial Trust** environments and requires the **Full Trust** permission level.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.GetImages Overload List](#)

GetMarkupAnnotationData Method

Retrieves all text markup annotations from a page in a PDF document.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetMarkupAnnotationData(  
    ByVal pageNumber As Integer  
) As IList(of PdfMarkupAnnotationData)
```

C#

```
public IList<PdfMarkupAnnotationData> GetMarkupAnnotationData(  
    int pageNumber  
)
```

Parameters

pageNumber
An integer value that specifies the number of a page where the markup annotations are located.

Return Value

A collection of DevExpress.Pdf.PdfMarkupAnnotationData objects that represent markup annotation data in a page.

Remarks

The **GetMarkupAnnotationData** method returns only text markup annotation data in a page

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T545429>.

This example shows how to change an existing markup annotation's settings in a PDF document.
To retrieve all text markup annotations in a page, call the [GetMarkupAnnotationData](#) method.

C#

```

(Program.cs)
using DevExpress.Pdf;
using System;
using System.Collections.Generic;
namespace ModifyExistingMarkupAnnotation {
    class Program {
        static void Main(string[] args) {
            // Create a PDF Document Processor.
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document.
                processor.LoadDocument("../\\..\\Document.pdf");
                // Get a list of annotations in the first document page.
                IList<PdfMarkupAnnotationData> annotations = processor.GetMarkupAnnotationData(1);
                if (annotations.Count > 0) {
                    // Change the properties of the first annotation.
                    annotations[0].Author = "Bill Smith";
                    annotations[0].Contents = "Important!";
                    annotations[0].Color = new PdfRgbColor(0.6, 0.7, 0.3);
                    annotations[0].Opacity = 0.2;
                    // Save the result document.
                    processor.SaveDocument("../\\..\\Result.pdf");
                }
                else
                    Console.WriteLine("The specified document page does not contain markup annotations!");
            }
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Collections.Generic
Namespace ModifyExistingMarkupAnnotation
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF Document Processor.
            Using processor As New PdfDocumentProcessor()
                ' Load a document.
                processor.LoadDocument("../\\..\\Document.pdf")
                ' Get a list of annotations in the first document page.
                Dim annotations As IList(Of PdfMarkupAnnotationData) = processor.GetMarkupAnnotationData(1)
                If annotations.Count > 0 Then
                    ' Change the properties of the first annotation.
                    annotations(0).Author = "Bill Smith"
                    annotations(0).Contents = "Important!"
                    annotations(0).Color = New PdfRgbColor(0.6, 0.7, 0.3)
                    annotations(0).Opacity = 0.2
                    ' Save the result document.
                    processor.SaveDocument("../\\..\\Result.pdf")
                Else
                    Console.WriteLine("The specified document page does not contain markup annotations!")
                End If
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)

GetPageText Method

Obtains text from the specified page.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetPageText(  
    ByVal pageNumber As Integer  
) As String
```

C#

```
public string GetPageText(  
    int pageNumber  
)
```

Parameters

pageNumber

An integer value that specifies the page number.

Return Value

A [System.String](#) value that represents text obtained from the specified page.

Remarks

If a document does not contain the specified page, the **GetPageText** method returns an empty string.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

GetText Method

Selects the text found within the specified document area.

Overload List

Name	Description
string GetText(PdfDocumentArea area)	Selects the text found within the specified document area.
string GetText(PdfDocumentPosition startPosition, PdfDocumentPosition endPosition)	Retrieves the text found within the specified document positions.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.GetText Overload List](#)

Selects the text found within the specified document area.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetText(  
    ByVal area As PdfDocumentArea  
) As String
```

C#

```
public string GetText(  
    PdfDocumentArea area  
)
```

Parameters

area
A DevExpress.Pdf.PdfDocumentArea object.

Return Value

A [System.String](#) value, specifying the text.

Remarks

The overloaded **GetText** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.GetText Overload List](#)

Retrieves the text found within the specified document positions.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetText(  
    ByVal startPosition As PdfDocumentPosition,  
    ByVal endPosition As PdfDocumentPosition  
) As String
```

C#

```
public string GetText(  
    PdfDocumentPosition startPosition,  
    PdfDocumentPosition endPosition  
)
```

Parameters*startPosition*

A DevExpress.Pdf.PdfDocumentPosition object that is the initial document position starting from which the text is obtained from a document.

endPosition

A DevExpress.Pdf.PdfDocumentPosition object that is the final document position starting from which the text is not obtained from a document.

Return Value

A [System.String](#) value, specifying the text.

Remarks

The overloaded **GetText** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.GetText Overload List](#)

Import Method

Imports interactive form data from the specified stream.

Overload List

Name	Description
void Import(Stream stream)	Imports interactive form data from the specified stream.
void Import(string fileName)	Imports interactive form data from the specified file.
void Import(string fileName, PdfFormDateFormat format)	Imports interactive form data from the specified file using the specified format.
void Import(Stream stream, PdfFormDateFormat format)	Imports interactive form data from the specified stream using the specified format.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.Import Overload List](#)

Imports interactive form data from the specified stream.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Import(  
    ByVal stream As Stream  
)
```

C#

```
public void Import(  
    Stream stream  
)
```

Parameters

stream
A [System.IO.Stream](#) class descendant specifying the stream containing the file with interactive form data.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.Import Overload List](#)

Imports interactive form data from the specified file.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Import(  
    ByVal fileName As String  
)
```

C#

```
public void Import(  
    string fileName  
)
```

Parameters

fileName

A [System.String](#) specifying the path to the file which contains interactive form data.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.Import Overload List](#)

Imports interactive form data from the specified file using the specified format.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Import(  
    ByVal fileName As String,  
    ByVal format As PdfFormDateFormat  
)
```

C#

```
public void Import(  
    string fileName,  
    PdfFormDateFormat format  
)
```

Parameters

fileName

A [System.String](#) specifying the path to the file with interactive form data.

format

A DevExpress.Pdf.PdfFormDateFormat enumeration value.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.Import Overload List](#)

Imports interactive form data from the specified stream using the specified format.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Import(  
    ByVal stream As Stream,  
    ByVal format As PdfFormDateFormat  
)
```

C#

```
public void Import(  
    Stream stream,  
    PdfFormDateFormat format  
)
```

Parameters

stream

A [System.IO.Stream](#) class descendant specifying the stream containing the file with interactive form data.

format

A DevExpress.Pdf.PdfFormDateFormat enumeration value.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.Import Overload List](#)

InsertNewPage Method

Inserts a new page with a specified page number and page size into the document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function InsertNewPage(  
    ByVal pageNumber As Integer,  
    ByVal mediaBox As PdfRectangle  
) As PdfPage
```

C#

```
public PdfPage InsertNewPage(  
    int pageNumber,  
    PdfRectangle mediaBox  
)
```

Parameters

pageNumber

An integer value that is the number of a page that should be inserted into the document.

mediaBox

A DevExpress.Pdf.PdfRectangle object that is the actual page size.

Return Value

A DevExpress.Pdf.PdfPage object that is the new inserted page.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

LoadDocument Method

Opens a PDF document from the specified file.

Overload List

Name	Description
void LoadDocument(string path)	Opens a PDF document from the specified file.
void LoadDocument(Stream stream)	Opens a PDF document from the specified stream.
void LoadDocument(string path, bool detachStreamAfterLoadComplete)	Opens a PDF document from the specified file using detach stream mode.
void LoadDocument(Stream stream, bool detachStreamAfterLoadComplete)	Opens a PDF document from the specified stream using detach stream mode.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.LoadDocument Overload List](#)

Opens a PDF document from the specified file.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub LoadDocument(  
    ByVal path As String  
)
```

C#

```
public void LoadDocument(  
    string path  
)
```

Parameters

path

A [System.String](#) value, specifying the file path.

Remarks

Since the **LoadDocument** method uses the *detachStreamAfterLoadComplete* parameter set to **false**, the PDF Document API component will lock a PDF file until it is loaded (this behavior is similar to that of Adobe® Reader®).

To unlock the PDF file, call another overloaded **LoadDocument** method with the *detachStreamAfterLoadComplete* parameter enabled.

Example

Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T210253 .	

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the `DevExpress.Pdf.PdfFormData.Value` property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#

```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic

```

(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.LoadDocument Overload List](#)

Opens a PDF document from the specified stream.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub LoadDocument(
    ByVal stream As Stream
)

```

C#

```

public void LoadDocument(
    Stream stream
)

```

Parameters

stream

A [System.IO.Stream](#) object.

Remarks

When loading a document, the [PDF Document API](#) component expects that the input stream will not be modified or closed before the component finishes using a document (the *detachStreamAfterLoadComplete* parameter is set to **false** in this method).

To force the PDF Document API component to complete all input operations after loading a document, call another overloaded **LoadDocument** method with the *detachStreamAfterLoadComplete* parameter enabled.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the `DevExpress.Pdf.PdfFormData.Value` property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#

```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic

```

(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.LoadDocument Overload List](#)

Opens a PDF document from the specified file using detach stream mode.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub LoadDocument(
    ByVal path As String,
    ByVal detachStreamAfterLoadComplete As Boolean
)

```

C#

```

public void LoadDocument(
    string path,
    bool detachStreamAfterLoadComplete
)

```

Parameters

path

A [System.String](#) value, specifying the file path.

detachStreamAfterLoadComplete

true if the PDF Document API component unlocks a file until it is loaded; otherwise **false**.

Remarks

The PDF Document API component locks a file until it is loaded by default (the *detachStreamAfterLoadComplete* parameter is set to **false**). If you do not want the PDF Document API component to lock the file, call the overloaded **LoadDocument** method with the *detachStreamAfterLoadComplete* parameter enabled.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the `DevExpress.Pdf.PdfFormData.Value` property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#

```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic

```

(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.LoadDocument Overload List](#)

[DevExpress.XtraPdfViewer.PdfViewer.DetachStreamAfterLoadComplete](#)

Opens a PDF document from the specified stream using detach stream mode.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub LoadDocument(
    ByVal stream As Stream,
    ByVal detachStreamAfterLoadComplete As Boolean
)

```

C#

```

public void LoadDocument(
    Stream stream,
    bool detachStreamAfterLoadComplete
)

```

Parameters

stream

A [System.IO.Stream](#) object.

detachStreamAfterLoadComplete


true the PDF Document API component completes all input operations after loading a document; **false** the stream should not be closed or modified until the PDF Document API component finishes using a document.

Remarks

When loading a document, the [PDF Document API](#) component expects that the input stream will not be modified or closed before the component finishes using a document (the *detachStreamAfterLoadComplete* parameter is set to **false** by default).

If you want to close or modify the stream after a document is loaded to the PDF Document API component, call the overloaded **LoadDocument** method with the *detachStreamAfterLoadComplete* parameter enabled.

Example

 Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T210253 .	

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the *DevExpress.Pdf.PdfFormData.Value* property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#	
<pre>(PdfFormFilling.cs) // Load a document with an interactive form. using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) { documentProcessor.LoadDocument(filePath + fileName + ".pdf"); // Obtain interactive form data from a document. PdfFormData formData = documentProcessor.GetFormData(); // Specify the value for FirstName and LastName text boxes. formData["FirstName"].Value = "Janet"; formData["LastName"].Value = "Leverling"; // Specify the value for the Gender radio group. formData["Gender"].Value = "Female"; // Specify the check box checked appearance name. formData["Check"].Value = "Yes"; // Specify values for the Category list box. formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" }; // Obtain data from the Address form field and specify values for Address child form fields. PdfFormData address = formData["Address"]; // Specify the value for the Country combo box. address["Country"].Value = "United States"; // Specify the value for City and Address text boxes. address["City"].Value = "California"; address["Address"].Value = "20 Maple Avenue"; // Apply data to the interactive form. documentProcessor.ApplyFormData(formData); // Save the modified document. documentProcessor.SaveDocument(filePath + fileName + "_new.pdf"); btnFillFormData.Enabled = false; btnLoadFilledPDF.Enabled = true; }</pre>	

Visual Basic	
---------------------	--

```
(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.LoadDocument Overload List](#)

[DevExpress.XtraPdfViewer.PdfViewer.DetachStreamAfterLoadComplete](#)

NextWord Method

Returns the next word in a PDF document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function NextWord() As PdfPageWord
```

C#

```
public PdfPageWord NextWord()
```

Return Value

A [PdfPageWord](#) object, providing information about the document page corresponding to the specified word.

Remarks

The **NextWord** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PrevWord](#)

[PdfPageWord.PageNumber](#)

PrevWord Method

Returns the previous word in a PDF document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function PrevWord() As PdfPageWord
```

C#

```
public PdfPageWord PrevWord()
```

Return Value

A [PdfPageWord](#) object, providing information about the document page corresponding to the specified PDF word.

Remarks

The overloaded **PrevWord** method uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[NextWord](#)

[PdfPageWord.PageNumber](#)

Print Method

Prints the current document, using the specified [PDF printer settings](#).

Overload List

Name	Description
void Print(PdfPrinterSettings printerSettings)	Prints the current document, using the specified PDF printer settings .
void Print(PrinterSettings printerSettings)	Obsolete. Prints the current document, using the specified settings.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.Print Overload List](#)

Prints the current document, using the specified [PDF printer settings](#).

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Print(  
    ByVal printerSettings As PdfPrinterSettings  
)
```

C#

```
public void Print(  
    PdfPrinterSettings printerSettings  
)
```

Parameters

printerSettings
A DevExpress.Pdf.PdfPrinterSettings object.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T172274>.

This example shows how to print a document with custom printer settings.

C#

```
(Program.cs)
// Developer Express Code Central Example:
// How to use the PDF printer settings
using DevExpress.Pdf;
namespace PdfProcessorPrinterOptions {
    class Program {
        static void Main(string[] args) {
            // Create a Pdf Document Processor instance and load a PDF into it.
            PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor();
            documentProcessor.LoadDocument(@"..\..\Demo.pdf");
            // Declare the PDF printer settings.
            PdfPrinterSettings pdfPrinterSettings = new PdfPrinterSettings();
            // Specify the PDF printer settings.
            pdfPrinterSettings.PageOrientation = PdfPrintPageOrientation.Portrait;
            pdfPrinterSettings.PageNumbers = new int[] { 1, 3, 4, 5 };
            // Setting the PdfPrintScaleMode property to CustomScale requires
            // specifying the Scale property, as well.
            pdfPrinterSettings.ScaleMode = PdfPrintScaleMode.CustomScale;
            pdfPrinterSettings.Scale = 90;
            // Print the document using the specified printer settings.
            documentProcessor.Print(pdfPrinterSettings);
        }
    }
}
```

Visual Basic

```
(Program.vb)
' Developer Express Code Central Example:
' How to use the PDF printer settings
Imports DevExpress.Pdf
Namespace PdfProcessorPrinterOptions
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a Pdf Document Processor instance and load a PDF into it.
            Dim documentProcessor As New PdfDocumentProcessor()
            documentProcessor.LoadDocument("../..\\Demo.pdf")
            ' Declare the PDF printer settings.
            Dim pdfPrinterSettings As New PdfPrinterSettings()
            ' Specify the PDF printer settings.
            pdfPrinterSettings.PageOrientation = PdfPrintPageOrientation.Portrait
            pdfPrinterSettings.PageNumbers = New Integer() { 1, 3, 4, 5 }
            ' Setting the PdfPrintScaleMode property to CustomScale requires
            ' specifying the Scale property, as well.
            pdfPrinterSettings.ScaleMode = PdfPrintScaleMode.CustomScale
            pdfPrinterSettings.Scale = 90
            ' Print the document using the specified printer settings.
            documentProcessor.Print(pdfPrinterSettings)
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.Print Overload List](#)
[How to: Use the PDF Printer Settings](#)

NOTE: This member is now obsolete.

Use the Print(PdfPrinterSettings pdfPrinterSettings) overload of this method instead.

Obsolete. Prints the current document, using the specified settings.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Print(  
    ByVal printerSettings As PrinterSettings  
)
```

C#

```
public void Print(  
    PrinterSettings printerSettings  
)
```

Parameters

printerSettings

A System.Drawing.Printing.PrinterSettings object.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.Print Overload List](#)

[Print](#)

RemoveForm Method

Removes all interactive form fields from a PDF document.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Function RemoveForm() As [Boolean](#)

C#

public [bool](#) RemoveForm()

Return Value

If form fields were removed from a PDF document, the value is **true**; otherwise, **false**.

Remarks

The **RemoveForm** method returns **true** if an interactive form is successfully removed from a PDF document. If a PDF document does not contain an interactive form, this method returns **false**.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494240>.

This example shows how to remove a particular form field and a whole interactive form from a PDF document.

To remove all form fields from a document, call the [RemoveForm](#) method.

If the interactive form is removed successfully, this method returns **true**. If the document doesn't contain an interactive form that should be removed, this method returns **false**.

To remove a particular form field by its name, call the [RemoveFormField](#) method and pass a field name as an argument to this method.

The [RemoveFormField](#) method tries to find the field in a document using the field name. If this field is found in the document, then it will be removed from a document and this method returns **true**; otherwise, it returns **false**.

C#

```

(Program.cs)
using DevExpress.Pdf;
using System;
namespace RemoveInteractiveForm {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document with an interactive form.
                processor.LoadDocument("../..\\InteractiveForm.pdf");
                // Remove a form field by its name.
                if (processor.RemoveFormField("FirstName"))
                    // Save the modified document.
                    processor.SaveDocument("../..\\Result1.pdf");
                else
                    // Show a message if the form field was not found in a document.
                    Console.WriteLine("The form field was not removed from a document. Make sure, the form field name is correct.");
                // Remove the whole interactive form.
                if (processor.RemoveForm())
                    // Save the modified document.
                    processor.SaveDocument("../..\\Result2.pdf");
                else
                    // Show a message if the interactive form was not found in a document.
                    Console.WriteLine("The interactive form was not removed from a document. Make sure the document contains an interactive form.");
            }
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Namespace RemoveInteractiveForm
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document with an interactive form.
                processor.LoadDocument("../..\\InteractiveForm.pdf")
                ' Remove a form field by its name.
                If processor.RemoveFormField("FirstName") Then
                    ' Save the modified document.
                    processor.SaveDocument("../..\\Result1.pdf")
                Else
                    ' Show a message if the form field was not found in a document.
                    Console.WriteLine("The form field was not removed from a document. Make sure, the form field name is correct.")
                End If
                ' Remove the whole interactive form.
                If processor.RemoveForm() Then
                    ' Save the modified document.
                    processor.SaveDocument("../..\\Result2.pdf")
                Else
                    ' Show a message if the interactive form was not found in a document.
                    Console.WriteLine("The interactive form was not removed from a document. Make sure the document contains an interactive form.")
                End If
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)

RemoveFormField Method

Removes an interactive form field from a document using the field name.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function RemoveFormField(  
    ByVal name As String  
) As Boolean
```

C#

```
public bool RemoveFormField(  
    string name  
)
```

Parameters

name

A [System.String](#) which specifies the name of form field.

Return Value

If the form field was removed from a PDF document, the value is **true**; otherwise, **false**.

Remarks

The **RemoveFormField** method tries to find the field in a document using the field name. If this field is found in the document, then it will be removed from a document and this method returns **true**; otherwise, it returns **false**.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T494240>.

This example shows how to remove a particular form field and a whole interactive form from a PDF document.

To remove all form fields from a document, call the [RemoveForm](#) method.

If the interactive form is removed successfully, this method returns **true**. If the document doesn't contain an interactive form that should be removed, this method returns **false**.

To remove a particular form field by its name, call the [RemoveFormField](#) method and pass a field name as an argument to this method.

The [RemoveFormField](#) method tries to find the field in a document using the field name. If this field is found in the document, then it will be removed from a document and this method returns **true**; otherwise, it returns **false**.

C#

```

(Program.cs)
using DevExpress.Pdf;
using System;
namespace RemoveInteractiveForm {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Load a document with an interactive form.
                processor.LoadDocument("../..\\InteractiveForm.pdf");
                // Remove a form field by its name.
                if (processor.RemoveFormField("FirstName"))
                    // Save the modified document.
                    processor.SaveDocument("../..\\Result1.pdf");
                else
                    // Show a message if the form field was not found in a document.
                    Console.WriteLine("The form field was not removed from a document. Make sure, the form field name is correct.");
                // Remove the whole interactive form.
                if (processor.RemoveForm())
                    // Save the modified document.
                    processor.SaveDocument("../..\\Result2.pdf");
                else
                    // Show a message if the interactive form was not found in a document.
                    Console.WriteLine("The interactive form was not removed from a document. Make sure the document contains an interactive form.");
            }
        }
    }
}

```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Namespace RemoveInteractiveForm
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Load a document with an interactive form.
                processor.LoadDocument("../..\\InteractiveForm.pdf")
                ' Remove a form field by its name.
                If processor.RemoveFormField("FirstName") Then
                    ' Save the modified document.
                    processor.SaveDocument("../..\\Result1.pdf")
                Else
                    ' Show a message if the form field was not found in a document.
                    Console.WriteLine("The form field was not removed from a document. Make sure, the form field name is correct.")
                End If
                ' Remove the whole interactive form.
                If processor.RemoveForm() Then
                    ' Save the modified document.
                    processor.SaveDocument("../..\\Result2.pdf")
                Else
                    ' Show a message if the interactive form was not found in a document.
                    Console.WriteLine("The interactive form was not removed from a document. Make sure the document contains an interactive form.")
                End If
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)

RenderNewPage Method

Adds a new page with specified page size and created graphics to a document.

Overload List

Name	Description
int RenderNewPage(PdfRectangle mediaBox, PdfGraphics graphics)	Adds a new page with specified page size and created graphics to a document.
int RenderNewPage(PdfRectangle mediaBox, PdfGraphics graphics, Single dpiX, Single dpiY)	Adds a new page with specified page size, DPI, and created graphics to a document.

See Also
[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.RenderNewPage Overload List](#)

Adds a new page with specified page size and created graphics to a document.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function RenderNewPage(  
    ByVal mediaBox As PdfRectangle,  
    ByVal graphics As PdfGraphics  
) As Integer
```

C#

```
public int RenderNewPage(  
    PdfRectangle mediaBox,  
    PdfGraphics graphics  
)
```

Parameters

mediaBox
A DevExpress.Pdf.PdfRectangle object that is the page size, in points (1/72 of an inch).
graphics
A DevExpress.Pdf.PdfGraphics object that contains all graphics content that allows an application to draw on the page.

Return Value


An integer value, specifying the page number. The page number is started from **1**.

Remarks

The overloaded **RenderNewPage** method automatically converts world coordinates to page coordinates. See [Coordinate Systems](#) to learn more.

Use this method to add a new page with graphics content represented by the DevExpress.Pdf.PdfGraphics object to a PDF document.

Example

 Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T244516 .	

The PDF Document Processor allows you to draw graphic primitives on the PDF page, such as lines, Bezier curves, polygons, ellipses, paths, an image and a string.

This example shows how to draw a business card in the document foreground. To accomplish this task, do the following:

- Create a PDF document processor and add an empty document to it by calling the [CreateEmptyDocument](#) method.
- Create PDF graphics represented by an instance of the `DevExpress.Pdf.PdfGraphics` class by calling the [CreateGraphics](#) method. To access `DevExpress.Pdf.PdfGraphics`, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- To draw an image on the PDF page, call the `DevExpress.Pdf.PdfGraphics.DrawImage` method for the specified image at the specified location with the specified size.
- To draw text, call the `DevExpress.Pdf.PdfGraphics.DrawString` method at the specified location with the specified **Brush** and **Font** objects.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer.");
            }
        }
    }
}
```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.RenderNewPage Overload List](#)

Adds a new page with specified page size, DPI, and created graphics to a document.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Function RenderNewPage(
    ByVal mediaBox As PdfRectangle,
    ByVal graphics As PdfGraphics,
    ByVal dpiX As Single,
    ByVal dpiY As Single
) As Integer

```

C#

```

public int RenderNewPage(
    PdfRectangle mediaBox,
    PdfGraphics graphics,
    Single dpiX,
    Single dpiY
)

```

Parameters

mediaBox

A DevExpress.Pdf.PdfRectangle object that is the page size, in points (1/72 of an inch).

graphics

A DevExpress.Pdf.PdfGraphics object that contains all graphics content that allows an application to draw on the page.

dpiX

A [System.Single](#) object that represents the value, in dots per inch, for the horizontal resolution.

dpiY

A [System.Single](#) object that represents the value, in dots per inch, for the vertical resolution.

Return Value

An integer value, specifying the page number.

Remarks

The overloaded **RenderNewPage** method automatically converts world coordinates to page coordinates. See [Coordinate Systems](#) to learn more.

Use this method to add a new page with the specified DPI and the graphics content represented by the DevExpress.Pdf.PdfGraphics object to a PDF document.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T244516>.

The PDF Document Processor allows you to draw graphic primitives on the PDF page, such as lines, Bezier curves, polygons, ellipses, paths, an image and a string.

This example shows how to draw a business card in the document foreground. To accomplish this task, do the following:

- Create a PDF document processor and add an empty document to it by calling the [CreateEmptyDocument](#) method.
- Create PDF graphics represented by an instance of the DevExpress.Pdf.PdfGraphics class by calling the [CreateGraphics](#) method. To access DevExpress.Pdf.PdfGraphics, you need to reference the **DevExpress.Pdf.Drawing** assembly.
- To draw an image on the PDF page, call the DevExpress.Pdf.PdfGraphics.DrawImage method for the specified image at the specified location with the specified size.
- To draw text, call the DevExpress.Pdf.PdfGraphics.DrawString method at the specified location with the specified **Brush** and **Font** objects.
- Render a page with created graphics by calling the [RenderNewPage](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System;
using System.Drawing;
namespace DocumentCreationAPI {
    class Program {
        static void Main(string[] args) {
            using (PdfDocumentProcessor processor = new PdfDocumentProcessor()) {
                // Create an empty document.
                processor.CreateEmptyDocument("..\\..\\Result.pdf");
                // Create and draw PDF graphics.
                using (PdfGraphics graph = processor.CreateGraphics()) {
                    DrawGraphics(graph);
                    // Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph);
                }
            }
        }
        static void DrawGraphics(PdfGraphics graph) {
            // Draw text lines on the page.
            SolidBrush black = (SolidBrush)Brushes.Black;
            using (Font font1 = new Font("Times New Roman", 32, FontStyle.Bold)) {
                graph.DrawString("PDF Document Processor", font1, black, 180, 150);
            }
            using (Font font2 = new Font("Arial", 20)) {
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230);
            }
            using (Font font3 = new Font("Arial", 10)) {
                graph.DrawString("The PDF Document Processor is a non-visual component " +
                                "that provides the application programming interface of the PDF Viewer."
                                );
            }
        }
    }
}
```

Visual Basic

```
(Program.vb)
Imports DevExpress.Pdf
Imports System
Imports System.Drawing
Namespace DocumentCreationAPI
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            Using processor As New PdfDocumentProcessor()
                ' Create an empty document.
                processor.CreateEmptyDocument("../..\Result.pdf")
                ' Create and draw PDF graphics.
                Using graph As PdfGraphics = processor.CreateGraphics()
                    DrawGraphics(graph)
                    ' Render a page with graphics.
                    processor.RenderNewPage(PdfPaperSize.Letter, graph)
                End Using
            End Using
        End Sub
        Private Shared Sub DrawGraphics(ByVal graph As PdfGraphics)
            ' Draw text lines on the page.
            Dim black As SolidBrush = CType(Brushes.Black, SolidBrush)
            Using font1 As New Font("Times New Roman", 32, FontStyle.Bold)
                graph.DrawString("PDF Document Processor", font1, black, 180, 150)
            End Using
            Using font2 As New Font("Arial", 20)
                graph.DrawString("Display, Print and Export PDF Documents", font2, black, 168, 230)
            End Using
            Using font3 As New Font("Arial", 10)
                graph.DrawString("The PDF Document Processor is a non-visual component " & "that provides
            End Using
        End Sub
    End Class
End Namespace
```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.RenderNewPage Overload List](#)

ResetFormData Method

Resets all fields of the interactive form to their default values.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ResetFormData()
```

C#

```
public void ResetFormData()
```

Remarks

The default values are specified inside the interactive form in a document.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

SaveDocument Method

Saves the current document to the specified file path.

Overload List

Name	Description
void SaveDocument(string path)	Saves the current document to the specified file path.
void SaveDocument(Stream stream)	Saves the current document to the specified file stream.
void SaveDocument(string path, bool detachStream)	Saves the current PDF document to the specified file using the detach stream mode.
void SaveDocument(Stream stream, bool detachStream)	Saves the current PDF document to the specified stream using the detach stream mode.
void SaveDocument(Stream stream, PdfSaveOptions options)	Saves the current PDF document to the specified stream with encryption settings and document signature.
void SaveDocument(string path, PdfSaveOptions options)	Saves the current PDF document to the specified file path with encryption and sign settings.
void SaveDocument(string path, PdfSaveOptions options, bool detachStream)	Saves the current PDF document to the specified file with encryption settings and document signature using the detach stream mode.
void SaveDocument(Stream stream, PdfSaveOptions options, bool detachStream)	Saves the current PDF document to the specified stream with encryption settings and document signature using detach stream mode.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.SaveDocument Overload List](#)

Saves the current document to the specified file path.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SaveDocument(  
    ByVal path As String  
)
```

C#

```
public void SaveDocument(  
    string path  
)
```

Parameters

path

A [System.String](#) value, specifying the location of the saved document.

Remarks

The PDF Document API component locks a file while a document is saved (since the **SaveDocument** method uses the **detachStream** parameter set to **false**). To unlock the file, call another overloaded **SaveDocument** method with the **detachStream** parameter enabled.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the `DevExpress.Pdf.PdfFormData.Value` property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#

```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic

```

(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.SaveDocument Overload List](#)

Saves the current document to the specified file stream.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub SaveDocument(
    ByVal stream As Stream
)

```

C#

```

public void SaveDocument(
    Stream stream
)

```

Parameters

stream

A [System.IO.Stream](#) value, specifying the location of the saved document.

Remarks

The **SaveDocument** method expects the input stream will not be closed or modified while a PDF document is saved (the **detachStream** parameter is set to **false** in this method). Be aware that disposing of an output stream that has not been detached may cause errors on an attempt to apply further changes to a document. If you want to close the stream when a document is saved, call another overloaded **SaveDocument** method with the **detachStream** parameter enabled.

Important

Using the same stream for loading and saving a document may cause unpredictable results since the **detachStream** parameter is set to **false** in the **SaveDocument** method.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T210253>.

The example shows how to fill an existing interactive form using field names.

To obtain interactive form data, call the [GetFormData](#) method. Then, specify a value for a form field using the `DevExpress.Pdf.PdfFormData.Value` property.

To obtain the names of the interactive form fields, use the [GetFormFieldNames](#) method.

To apply data to the interactive form, call the [ApplyFormData](#) method.

To learn how to get a checked appearance name for the check box, see the [How to: Obtain a Checked Appearance Name for a Check Box](#) example.

To learn how to get a checked appearance name for the radio button, see the [How to: Obtain a Checked Appearance Name for Each Radio Button in the Radio Group](#) example.

C#

```
(PdfFormFilling.cs)
// Load a document with an interactive form.
using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
    documentProcessor.LoadDocument(filePath + fileName + ".pdf");
    // Obtain interactive form data from a document.
    PdfFormData formData = documentProcessor.GetFormData();
    // Specify the value for FirstName and LastName text boxes.
    formData["FirstName"].Value = "Janet";
    formData["LastName"].Value = "Leverling";
    // Specify the value for the Gender radio group.
    formData["Gender"].Value = "Female";
    // Specify the check box checked appearance name.
    formData["Check"].Value = "Yes";
    // Specify values for the Category list box.
    formData["Category"].Value = new string[] { "Entertainment", "Meals", "Morale" };
    // Obtain data from the Address form field and specify values for Address child form fields.
    PdfFormData address = formData["Address"];
    // Specify the value for the Country combo box.
    address["Country"].Value = "United States";
    // Specify the value for City and Address text boxes.
    address["City"].Value = "California";
    address["Address"].Value = "20 Maple Avenue";
    // Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData);
    // Save the modified document.
    documentProcessor.SaveDocument(filePath + fileName + "_new.pdf");
    btnFillFormData.Enabled = false;
    btnLoadFilledPDF.Enabled = true;
}
```

Visual Basic

```

(PdfFormFilling.vb)
' Load a document with an interactive form.
Using documentProcessor As New PdfDocumentProcessor()
    documentProcessor.LoadDocument(filePath & fileName & ".pdf")
    ' Obtain interactive form data from a document.
    Dim formData As PdfFormData = documentProcessor.GetFormData()
    ' Specify the value for FirstName and LastName text boxes.
    formData("FirstName").Value = "Janet"
    formData("LastName").Value = "Leverling"
    ' Specify the value for the Gender radio group.
    formData("Gender").Value = "Female"
    ' Specify the check box checked appearance name.
    formData("Check").Value = "Yes"
    ' Specify values for the Category list box.
    formData("Category").Value = New String() { "Entertainment", "Meals", "Morale" }
    ' Obtain data from the Address form field and specify values for Address child form fields.
    Dim address As PdfFormData = formData("Address")
    ' Specify the value for the Country combo box.
    address("Country").Value = "United States"
    ' Specify the value for City and Address text boxes.
    address("City").Value = "California"
    address("Address").Value = "20 Maple Avenue"
    ' Apply data to the interactive form.
    documentProcessor.ApplyFormData(formData)
    ' Save the modified document.
    documentProcessor.SaveDocument(filePath & fileName & "_new.pdf")
    btnFillFormData.Enabled = False
    btnLoadFilledPDF.Enabled = True
End Using

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.SaveDocument Overload List](#)

Saves the current PDF document to the specified file using the detach stream mode.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub SaveDocument(
    ByVal path As String,
    ByVal detachStream As Boolean
)

```

C#

```

public void SaveDocument(
    string path,
    bool detachStream
)

```

Parameters

path

A [System.String](#) that is the file path to which a document is saved.

detachStream

true, the PDF Document API component unlocks a file while a document is saved; **false**, the PDF Document API component locks a file while a document is saved (default mode).

Remarks

The PDF Document API component locks a file while a document is saved (the **detachStream** is set to **false** by default). To unlock the file, call the overloaded **SaveDocument** method with the **detachStream** parameter enabled.

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.SaveDocument Overload List](#)

Saves the current PDF document to the specified stream using the detach stream mode.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub SaveDocument(  
    ByVal stream As Stream,  
    ByVal detachStream As Boolean  
)
```

C#

```
public void SaveDocument(  
    Stream stream,  
    bool detachStream  
)
```

Parameters

stream

A [System.IO.Stream](#) object that is the stream to which the PDF document is saved.

detachStream

true, the PDF Document API component completes all operations after saving a document; **false**, the output stream should not be modified or closed while a PDF document is saved.

Remarks

When saving a document, the PDF Document API component expects that the stream will not be closed or modified before the control finishes using a document (the *detachStream* parameter is set to **false** by default). Be aware that disposing of an output stream that has not been detached may cause errors on an attempt to apply further changes to a document. If you want to close the stream when a document is saved, call the overloaded **SaveDocument** method with the **detachStream** parameter enabled.

 **Important**

Using the same stream for loading and saving a document may cause unpredictable results if the **detachStream** parameter is set to **false**.

See Also

[PdfDocumentProcessor Class](#)
[PdfDocumentProcessor Members](#)
[DevExpress.Pdf Namespace](#)
[PdfDocumentProcessor.SaveDocument Overload List](#)

Saves the current PDF document to the specified stream with encryption settings and document signature.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub SaveDocument(  
    ByVal stream As Stream,  
    ByVal options As PdfSaveOptions  
)
```

C#

```
public void SaveDocument(  
    Stream stream,  
    PdfSaveOptions options  
)
```

Parameters

stream
A [System.IO.Stream](#) value, specifying the location of the saved document.
options
A DevExpress.Pdf.PdfSaveOptions that contains the encryption settings and document signature that should be saved.

Remarks

The **SaveDocument** method expects the input stream will not be closed or modified while a PDF document is saved (the **detachStream** parameter is set to **false** in this method). Be aware that disposing of an output stream that has not been detached may cause errors on an attempt to apply further changes to a document. If you want to close the stream when a document is saved, call another overloaded **SaveDocument** method with the **detachStream** parameter enabled.

Important

Using the same stream for loading and saving a document may cause unpredictable results, since the **detachStream** parameter is set to **false** in this method.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T243905>.

This example illustrates how to apply a digital signature to a PDF document.

To accomplish this task, do the following:

- Create a PDF Document API component represented by an instance of the [PdfDocumentProcessor](#) class.
- Load a document from a file using the [LoadDocument](#) method.
- Create a certificate using a certificate file name and a password to access the certificate.
- Create a document digital signature represented by a DevExpress.Pdf.PdfSignature object using the certificate;
- Specify signing location, contact info and reason using the DevExpress.Pdf.PdfSignature.Location, DevExpress.Pdf.PdfSignature.ContactInfo and DevExpress.Pdf.PdfSignature.Reason properties, respectively.
- Save the signed document with signing information by calling the [SaveDocument](#) method.

C#

```
(Program.cs)
using DevExpress.Pdf;
using System.Security.Cryptography.X509Certificates;
namespace PDFSignature {
    class Program {
        static void Main(string[] args) {
            // Create a PDF document processor.
            using (PdfDocumentProcessor documentProcessor = new PdfDocumentProcessor()) {
                // Load a PDF document.
                documentProcessor.LoadDocument(@"..\..\Demo.pdf");
                // Load a certificate from a file.
                X509Certificate2 cert = new X509Certificate2(@"..\..\SignDemo.pfx", "dxdemo");
                // Create a PDF signature and specify signing location, contact info and reason.
                PdfSignature signature = new PdfSignature(cert) {
                    Location = "Location",
                    ContactInfo = "ContactInfo",
                    Reason = "Reason"
                };
                // Save the signed document.
                documentProcessor.SaveDocument(@"..\..\SignedDocument.pdf", new PdfSaveOptions() { Signatu
            }
        }
    }
}
```

Visual Basic

```

(Program.vb)
Imports DevExpress.Pdf
Imports System.Security.Cryptography.X509Certificates
Namespace PDFSignature
    Friend Class Program
        Shared Sub Main(ByVal args() As String)
            ' Create a PDF document processor.
            Using documentProcessor As New PdfDocumentProcessor()
                ' Load a PDF document.
                documentProcessor.LoadDocument("../..\Demo.pdf")
                ' Load a certificate from a file.
                Dim cert As New X509Certificate2("../..\SignDemo.pfx", "dxdemo")
                ' Create a PDF signature and specify signing location, contact info and reason.
                Dim signature As New DevExpress.Pdf.PdfSignature(cert) With {.Location = "Location", .ContactInfo = "Contact Info", .Reason = "Reason"}
                ' Save the signed document.
                documentProcessor.SaveDocument("../..\SignedDocument.pdf", New PdfSaveOptions() With {.Signature = signature})
            End Using
        End Sub
    End Class
End Namespace

```

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.SaveDocument Overload List](#)

Saves the current PDF document to the specified file path with encryption and sign settings.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

Public Sub SaveDocument(
    ByVal path As String,
    ByVal options As PdfSaveOptions
)

```

C#

```

public void SaveDocument(
    string path,
    PdfSaveOptions options
)

```

Parameters

path

A [System.String](#), specifying the path to the directory to which the PDF document should be saved.

options

A DevExpress.Pdf.PdfSaveOptions that contains the encryption and sign settings of a PDF document that should be saved.

Remarks

The PDF Document API component locks a file while a document is saved (the **SaveDocument** method uses the **detachStream** parameter set to **false**). To unlock the file, call another overloaded **SaveDocument** method with the **detachStream** parameter enabled.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.SaveDocument Overload List](#)

Saves the current PDF document to the specified file with encryption settings and document signature using the detach stream mode.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SaveDocument(  
    ByVal path As String,  
    ByVal options As PdfSaveOptions,  
    ByVal detachStream As Boolean  
)
```

C#

```
public void SaveDocument(  
    string path,  
    PdfSaveOptions options,  
    bool detachStream  
)
```

Parameters

path

A [System.String](#) that is the file path to which a document is saved.

options

A DevExpress.Pdf.PdfSaveOptions object that contains the encryption settings and document signature that should be saved.

detachStream

true, the PDF Document API component unlocks a file while a document is saved; **false**, the PDF Document API component locks a file while a document is saved (default mode).

Remarks

The PDF Document API component locks a file while a document is saved (the **detachStream** is set to **false** by default). To unlock the file, call the **SaveDocument** method with the **detachStream** parameter enabled.

See Also

[PdfDocumentProcessor Class](#)

[PdfDocumentProcessor Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.SaveDocument Overload List](#)

Saves the current PDF document to the specified stream with encryption settings and document signature using detach stream mode.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SaveDocument(  
    ByVal stream As Stream,  
    ByVal options As PdfSaveOptions,  
    ByVal detachStream As Boolean  
)
```

C#

```
public void SaveDocument(  
    Stream stream,  
    PdfSaveOptions options,  
    bool detachStream  
)
```

Parameters

stream

A [System.IO.Stream](#) object that is the stream to which the PDF document is saved.

options

A DevExpress.Pdf.PdfSaveOptions object that contains the encryption settings and document signature that should be saved.

detachStream

true, the PDF Document API component completes all operations after saving a document; **false**, the stream should not be closed or modified while a PDF document is saved.

Remarks

The **SaveDocument** method expects the input stream will not be closed or modified while a PDF document is saved (the **detachStream** parameter is set to **false** by default). Be aware that disposing of an output stream that has not been detached may cause errors on an attempt to apply further changes to a document. If you want to close the stream when a document is saved, call the overloaded **SaveDocument** method with the **detachStream** parameter enabled.

 **Important**

Using the same stream for loading and saving a document may cause unpredictable results if the **detachStream** parameter is set to **false**.

See Also[PdfDocumentProcessor Class](#)[PdfDocumentProcessor Members](#)[DevExpress.Pdf Namespace](#)[PdfDocumentProcessor.SaveDocument Overload List](#)

PdfPageWord Class

An individual word corresponding to a specific PDF page.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class PdfPageWord Inherits PdfWord
```

C#

```
public class PdfPageWord : PdfWord
```

Remarks

The **PdfPageWord** class uses the page coordinate system. See the [Coordinate Systems](#) topic to learn more.

Use the [PageNumber](#) property of this class to obtain the page number corresponding to a specific word in a PDF.

In [PDF Document API](#), a **PdfPageWord** instance is returned by the [PdfDocumentProcessor.NextWord](#) and [PdfDocumentProcessor.PrevWord](#) methods.

Inheritance Hierarchy

Object

PdfWord

PdfPageWord

See Also

[PdfPageWord Members](#)






[DevExpress.Pdf Namespace](#)

PdfPageWord Members







An individual word corresponding to a specific PDF page.

The following tables list the members exposed by the [PdfPageWord](#) type.

Public Properties

	Name	Description
	Characters	Returns a list of the word's characters. (Inherited from PdfWord)
	PageNumber	Indicates the page number corresponding to a specific word in a PDF file.
	Rectangles	Returns rectangles surrounding the word. (Inherited from PdfWord)
	Segments	Returns a list of the word's segments. (Inherited from PdfWord)
	Text	Returns a Unicode representation of the word's characters. (Inherited from PdfWord)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also






[PdfPageWord Members](#)
[DevExpress.Pdf Namespace](#)

PdfPageWord Properties







An individual word corresponding to a specific PDF page.

The following tables list the members exposed by the [PdfPageWord](#) type.

Public Properties

	Name	Description
	Characters	Returns a list of the word's characters. (Inherited from PdfWord)
	PageNumber	Indicates the page number corresponding to a specific word in a PDF file.
	Rectangles	Returns rectangles surrounding the word. (Inherited from PdfWord)
	Segments	Returns a list of the word's segments. (Inherited from PdfWord)
	Text	Returns a Unicode representation of the word's characters. (Inherited from PdfWord)

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[PdfPageWord Members](#)
[DevExpress.Pdf Namespace](#)

PageNumber Property

Indicates the page number corresponding to a specific word in a PDF file.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property PageNumber As Integer
```

C#

```
public int PageNumber { get; }
```

Property Value

An integer value, specifying the number of a PDF page.

See Also

[PdfPageWord Class](#)

[PdfPageWord Members](#)

[DevExpress.Pdf Namespace](#)

[PdfDocumentProcessor.PrevWord](#)

[PdfDocumentProcessor.NextWord](#)

PdfViewerExtensions Class

Defines extension methods that are used to extend the functionality of the WinForms PDF Viewer and WPF PDF Viewer.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
NotInheritable Public Class PdfViewerExtensions Inherits Object
```

C#

```
public sealed abstract class PdfViewerExtensions : object
```

Remarks

To access extension methods for the WinForms PDF Viewer and WPF PDF Viewer, add the **DevExpress.Docs** reference to your application.

Important

The Universal Subscription or an additional Document Server Subscription is required to use extension methods in production code. Please refer to the [DevExpress Subscription](#) page for pricing information.

The extension methods provide the following features for the WinForms PDF Viewer and WPF PDF Viewer.

Export/Import Interactive Form Data Programmatically

To export interactive form data from a document to one of the supported formats in the PDF Viewer, call one of the [Export](#) extension methods with the specified data format settings. See the [How to: Export AcroForm Data to XML](#) example.

To import interactive form data, call one of the [Import](#) extension methods with or without data format settings. See the [How to: Import Interactive Form Data from XML](#) example.

Protect a Document with a Password and Specify a Signature

To protect a document with an owner or user password in the PDF Viewer, call one of the [SaveDocument](#) extension methods and pass the `DevExpress.Pdf.PdfSaveOptions` object containing the encryption settings as a parameter. See the [Protecting a Document](#) topic to learn more.

To specify the document signature, call one of the [SaveDocument](#) extension methods and pass the `DevExpress.Pdf.PdfSaveOptions` object containing the document signature as a parameter. See the [Signing a Document](#) topic.

Inheritance Hierarchy

[Object](#)

PdfViewerExtensions

See Also

[PdfViewerExtensions Members](#)

[DevExpress.Pdf Namespace](#)

[How to: Export AcroForm Data to XML](#)




[How to: Import Interactive Form Data from XML](#)

PdfViewerExtensions Members

Defines extension methods that are used to extend the functionality of the WinForms PDF Viewer and WPF PDF Viewer.

The following tables list the members exposed by the [PdfViewerExtensions](#) type.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Export	Overloaded. Exports interactive form data to a specified stream using form data format. This is an extension method.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Import	Overloaded. Imports interactive form data from a specified stream using form data format. This is an extension method.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	SaveDocument	Overloaded. Saves the document to the specified stream with encryption settings and document signature. This is an extension method.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[PdfViewerExtensions Members](#)

[DevExpress.Pdf Namespace](#)

How to: Export AcroForm Data to XML





[How to: Import Interactive Form Data from XML](#)

PdfViewerExtensions Methods

Defines extension methods that are used to extend the functionality of the WinForms PDF Viewer and WPF PDF Viewer.

The following tables list the members exposed by the [PdfViewerExtensions](#) type.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Export	Overloaded. Exports interactive form data to a specified stream using form data format. This is an extension method.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Import	Overloaded. Imports interactive form data from a specified stream using form data format. This is an extension method.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	SaveDocument	Overloaded. Saves the document to the specified stream with encryption settings and document signature. This is an extension method.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[PdfViewerExtensions Members](#)

[DevExpress.Pdf Namespace](#)

How to: Export AcroForm Data to XML

[How to: Import Interactive Form Data from XML](#)

Export Method

Exports interactive form data to a specified stream using form data format. This is an extension method.

Overload List

Name	Description
static void Export(IPdfViewer viewer, Stream stream, PdfFormDateFormat format)	Exports interactive form data to a specified stream using form data format. This is an extension method.
static void Export(IPdfViewer viewer, string fileName, PdfFormDateFormat format)	Exports interactive form data to the file using the specified form data format. This is an extension method.

See Also
[PdfViewerExtensions Class](#)
[PdfViewerExtensions Members](#)
[DevExpress.Pdf Namespace](#)
[PdfViewerExtensions.Export Overload List](#)

Exports interactive form data to a specified stream using form data format. This is an extension method.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Export(  
    ByVal viewer As IPdfViewer,  
    ByVal stream As Stream,  
    ByVal format As PdfFormDateFormat  
)
```

C#

```
public static void Export(  
    IPdfViewer viewer,  
    Stream stream,  
    PdfFormDateFormat format  
)
```

Parameters

viewer
A DevExpress.XtraPdfViewer.PdfViewer or DevExpress.Xpf.PdfViewer.PdfViewerControl object that implements the DevExpress.Pdf.IPdfViewer interface.

stream
A [System.IO.Stream](#) value, containing the document to which interactive form data should be exported.

format
A DevExpress.Pdf.PdfFormDateFormat enumeration value that represents one of the supported formats for PDF form data values.

Remarks

See Export and Import of Interactive Form Data (WinForms PDF Viewer) and Export and Import of Interactive Form Data (WPF PDF Viewer) topics to learn more.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T273545>.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

This example shows how to export AcroForm data (interactive form data) from a PDF document to XML format.

You can also export the AcroForm data to FDF, XFDF, and TXT formats using the approach described below.

To export AcroForm to XML format:

- load a document containing interactive forms (e.g., from a file path) into the PDF Viewer using the `DevExpress.XtraPdfViewer.PdfViewer.LoadDocument` method;
- call one of the [Export](#) overloaded methods, for example, with a specified XML file name including a file path where the exported document will be located, and the XML data format.

Note

You may need to add the **DevExpress.Docs** reference to your application to access the **PdfViewer.Export** extension method.

C#

```
(Form1.cs)
using System.Windows.Forms;
using DevExpress.Pdf;
namespace ExportAcroFormDocument {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
            // Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\AcroForm.pdf");
            // Export the document to the xml format.
            pdfViewer1.Export("..\\..\\AcroForm.xml", PdfFormDataType.Xml);
        }
    }
}
```

Visual Basic

```
(Form1.vb)
Imports System.Windows.Forms
Imports DevExpress.Pdf
Namespace ExportAcroFormDocument
    Partial Public Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
            ' Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\AcroForm.pdf")
            ' Export the document to the xml format.
            pdfViewer1.Export("..\\..\\AcroForm.xml", PdfFormDataType.Xml)
        End Sub
    End Class
End Namespace
```

See Also

[PdfViewerExtensions Class](#)

[PdfViewerExtensions Members](#)

[DevExpress.Pdf Namespace](#)

[PdfViewerExtensions.Export Overload List](#)

Exports interactive form data to the file using the specified form data format. This is an extension method.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Export(  
    ByVal viewer As IPdfViewer,  
    ByVal fileName As String,  
    ByVal format As PdfFormDateFormat  
)
```

C#

```
public static void Export(  
    IPdfViewer viewer,  
    string fileName,  
    PdfFormDateFormat format  
)
```

Parameters

viewer
A DevExpress.XtraPdfViewer.PdfViewer or DevExpress.Xpf.PdfViewer.PdfViewerControl object that implements DevExpress.Pdf.IPdfViewer interface.

fileName
A [System.String](#), specifying the path to the file to which the document with interactive form data should be exported.

format
A DevExpress.Pdf.PdfFormDateFormat enumeration value that represents one of the supported formats for form data values.

Remarks

See Export and Import of Interactive Form Data (WinForms PDF Viewer) and Export and Import of Interactive Form Data (WPF PDF Viewer) topics to learn more.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T273545>.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

This example shows how to export AcroForm data (interactive form data) from a PDF document to XML format.

You can also export the AcroForm data to FDF, XFDF, and TXT formats using the approach described below.

To export AcroForm to XML format:

- load a document containing interactive forms (e.g., from a file path) into the PDF Viewer using the DevExpress.XtraPdfViewer.PdfViewer.LoadDocument method;
- call one of the [Export](#) overloaded methods, for example, with a specified XML file name including a file path where the exported document will be located, and the XML data format.

Note

You may need to add the **DevExpress.Docs** reference to your application to access the **PdfViewer.Export** extension method.

C#

```
(Form1.cs)
using System.Windows.Forms;
using DevExpress.Pdf;
namespace ExportAcroFormDocument {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
            // Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\AcroForm.pdf");
            // Export the document to the xml format.
            pdfViewer1.Export("..\\..\\AcroForm.xml", PdfFormDataType.Xml);
        }
    }
}
```

Visual Basic

```
(Form1.vb)
Imports System.Windows.Forms
Imports DevExpress.Pdf
Namespace ExportAcroFormDocument
    Partial Public Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
            ' Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\AcroForm.pdf")
            ' Export the document to the xml format.
            pdfViewer1.Export("..\\..\\AcroForm.xml", PdfFormDataType.Xml)
        End Sub
    End Class
End Namespace
```

See Also

[PdfViewerExtensions Class](#)

[PdfViewerExtensions Members](#)

[DevExpress.Pdf Namespace](#)

[PdfViewerExtensions.Export Overload List](#)

Import Method

Imports interactive form data from a stream. This is an extension method.

Overload List

Name	Description
static void Import(IPdfViewer viewer, Stream stream)	Imports interactive form data from a stream. This is an extension method.
static void Import(IPdfViewer viewer, string fileName)	Imports interactive form data from the file. This is an extension method.
static void Import(IPdfViewer viewer, Stream stream, PdfFormDateFormat format)	Imports interactive form data from a specified stream using form data format. This is an extension method.
static void Import(IPdfViewer viewer, string fileName, PdfFormDateFormat format)	Imports interactive form data from the specified file with the specified form data format. This is an extension method.

See Also
[PdfViewerExtensions Class](#)
[PdfViewerExtensions Members](#)
[DevExpress.Pdf Namespace](#)
[PdfViewerExtensions.Import Overload List](#)

Imports interactive form data from a stream. This is an extension method.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal viewer As IPdfViewer,  
    ByVal stream As Stream  
)
```

C#

```
public static void Import(  
    IPdfViewer viewer,  
    Stream stream  
)
```

Parameters

viewer
A DevExpress.XtraPdfViewer.PdfViewer or DevExpress.Xpf.PdfViewer.PdfViewerControl object that implements the DevExpress.Pdf.IPdfViewer interface.

stream
A [System.IO.Stream](#) value, specifying the stream containing document with interactive form data.

Remarks

See Export and Import of Interactive Form Data (WinForms PDF Viewer) and Export and Import of Interactive Form Data (WPF PDF Viewer) topics to learn more.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T273679>.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

This example demonstrates how to import AcroForm data (interactive form data) from XML format to a PDF document.

You can also import the AcroForm data from FDF, XFDF, and TXT formats, as described below.

To import interactive forms from XML format:

- load an interactive forms document (e.g., from a file path), in which the data will be imported, into the PDF Viewer using the `DevExpress.XtraPdfViewer.PdfViewer.LoadDocument` method;
- call one of the [Import](#) overloaded methods, for example, with a specified XML file that contains imported data.

Note

You may need to add the **DevExpress.Docs** reference to your application to access the **PdfViewer.Import** extension methods.

C#

```
(Form1.cs)
using System.Windows.Forms;
using DevExpress.Pdf;
namespace ImportAcroForm {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
            // Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\InitialAcroForm.pdf");
            // Import the document from an XML format.
            pdfViewer1.Import("..\\..\\FilledAcroForm.xml");
            // Save the imported document.
            pdfViewer1.SaveDocument("..\\..\\ImportedAcroForm.pdf");
        }
    }
}
```

Visual Basic

```
(Form1.vb)
Imports System.Windows.Forms
Imports DevExpress.Pdf
Namespace ImportAcroForm
    Partial Public Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
            ' Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\InitialAcroForm.pdf")
            ' Import the document from an XML format.
            pdfViewer1.Import("..\\..\\FilledAcroForm.xml")
            ' Save the imported document.
            pdfViewer1.SaveDocument("..\\..\\ImportedAcroForm.pdf")
        End Sub
    End Class
End Namespace
```

See Also

[PdfViewerExtensions Class](#)

[PdfViewerExtensions Members](#)

[DevExpress.Pdf Namespace](#)

[PdfViewerExtensions.Import Overload List](#)

Imports interactive form data from the file. This is an extension method.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic

```
static Public Sub Import(  
    ByVal viewer As IPdfViewer,  
    ByVal fileName As String  
)
```

C#

```
public static void Import(  
    IPdfViewer viewer,  
    string fileName  
)
```

Parameters

viewer
A DevExpress.XtraPdfViewer.PdfViewer or DevExpress.Xpf.PdfViewer.PdfViewerControl object that implements the DevExpress.Pdf.IPdfViewer interface.

fileName
A [System.String](#), specifying the path to the file from which interactive form data should be imported.

Remarks

See Export and Import of Interactive Form Data (WinForms PDF Viewer) and Export and Import of Interactive Form Data (WPF PDF Viewer) topics to learn more.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T273679>.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

This example demonstrates how to import AcroForm data (interactive form data) from XML format to a PDF document. You can also import the AcroForm data from FDF, XFDF, and TXT formats, as described below.

To import interactive forms from XML format:

- load an interactive forms document (e.g., from a file path), in which the data will be imported, into the PDF Viewer using the DevExpress.XtraPdfViewer.PdfViewer.LoadDocument method;
- call one of the [Import](#) overloaded methods, for example, with a specified XML file that contains imported data.

Note

You may need to add the **DevExpress.Docs** reference to your application to access the **PdfViewer.Import** extension methods.

C#

```
(Form1.cs)
using System.Windows.Forms;
using DevExpress.Pdf;
namespace ImportAcroForm {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
            // Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\InitialAcroForm.pdf");
            // Import the document from an XML format.
            pdfViewer1.Import("..\\..\\FilledAcroForm.xml");
            // Save the imported document.
            pdfViewer1.SaveDocument("..\\..\\ImportedAcroForm.pdf");
        }
    }
}
```

Visual Basic

```
(Form1.vb)
Imports System.Windows.Forms
Imports DevExpress.Pdf
Namespace ImportAcroForm
    Partial Public Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
            ' Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\InitialAcroForm.pdf")
            ' Import the document from an XML format.
            pdfViewer1.Import("..\\..\\FilledAcroForm.xml")
            ' Save the imported document.
            pdfViewer1.SaveDocument("..\\..\\ImportedAcroForm.pdf")
        End Sub
    End Class
End Namespace
```

See Also

[PdfViewerExtensions Class](#)

[PdfViewerExtensions Members](#)

[DevExpress.Pdf Namespace](#)

[PdfViewerExtensions.Import Overload List](#)

Imports interactive form data from a specified stream using form data format. This is an extension method.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(
    ByVal viewer As IPdfViewer,
    ByVal stream As Stream,
    ByVal format As PdfFormDateFormat
)
```

C#

```
public static void Import(
    IPdfViewer viewer,
    Stream stream,
    PdfFormDateFormat format
)
```

Parameters

viewer

A DevExpress.XtraPdfViewer.PdfViewer or DevExpress.Xpf.PdfViewer.PdfViewerControl object that implements the DevExpress.Pdf.IPdfViewer interface.

stream

A [System.IO.Stream](#) value, specifying the stream containing the document with interactive form data.

format

A DevExpress.Pdf.PdfFormDateFormat enumeration value that lists formats used to import the file with form data values.

Remarks

See Export and Import of Interactive Form Data (WinForms PDF Viewer) and Export and Import of Interactive Form Data (WPF PDF Viewer) topics to learn more.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T273679>.

Important

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

This example demonstrates how to import AcroForm data (interactive form data) from XML format to a PDF document.

You can also import the AcroForm data from FDF, XFDF, and TXT formats, as described below.

To import interactive forms from XML format:

- load an interactive forms document (e.g., from a file path), in which the data will be imported, into the PDF Viewer using the DevExpress.XtraPdfViewer.PdfViewer.LoadDocument method;
- call one of the [Import](#) overloaded methods, for example, with a specified XML file that contains imported data.

Note

You may need to add the **DevExpress.Docs** reference to your application to access the **PdfViewer.Import** extension methods.

C#

```
(Form1.cs)
using System.Windows.Forms;
using DevExpress.Pdf;
namespace ImportAcroForm {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
            // Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\InitialAcroForm.pdf");
            // Import the document from an XML format.
            pdfViewer1.Import("..\\..\\FilledAcroForm.xml");
            // Save the imported document.
            pdfViewer1.SaveDocument("..\\..\\ImportedAcroForm.pdf");
        }
    }
}
```

Visual Basic

```
(Form1.vb)
Imports System.Windows.Forms
Imports DevExpress.Pdf
Namespace ImportAcroForm
    Partial Public Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
            ' Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("../..\InitialAcroForm.pdf")
            ' Import the document from an XML format.
            pdfViewer1.Import("../..\FilledAcroForm.xml")
            ' Save the imported document.
            pdfViewer1.SaveDocument("../..\ImportedAcroForm.pdf")
        End Sub
    End Class
End Namespace
```

- See Also**
- [PdfViewerExtensions Class](#)
 - [PdfViewerExtensions Members](#)
 - [DevExpress.Pdf Namespace](#)
 - [PdfViewerExtensions.Import Overload List](#)

Imports interactive form data from the specified file with the specified form data format. This is an extension method.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal viewer As IPdfViewer,  
    ByVal fileName As String,  
    ByVal format As PdfFormDateFormat  
)
```

C#

```
public static void Import(  
    IPdfViewer viewer,  
    string fileName,  
    PdfFormDateFormat format  
)
```


Parameters

- viewer*
A DevExpress.XtraPdfViewer.PdfViewer or DevExpress.Xpf.PdfViewer.PdfViewerControl object that implements the DevExpress.Pdf.IPdfViewer interface.
- fileName*
A [System.String](#), specifying the path to the file from which the interactive form data should be imported.
- format*
A DevExpress.Pdf.PdfFormDateFormat enumeration value that lists formats for PDF form data values.

Remarks

See Export and Import of Interactive Form Data (WinForms PDF Viewer) and Export and Import of Interactive Form Data (WPF PDF Viewer) topics to learn more.

Example

 Show Me	
A complete sample project is available in the DevExpress Code Examples database at http://www.devexpress.com/example=T273679 .	

 **Important**

The Universal Subscription or an additional Office File API Subscription is required to use this example in production code. Refer to the [DevExpress Subscription](#) page for pricing information.

This example demonstrates how to import AcroForm data (interactive form data) from XML format to a PDF document.

You can also import the AcroForm data from FDF, XFDF, and TXT formats, as described below.

To import interactive forms from XML format:

- load an interactive forms document (e.g., from a file path), in which the data will be imported, into the PDF Viewer using the `DevExpress.XtraPdfViewer.PdfViewer.LoadDocument` method;
- call one of the [Import](#) overloaded methods, for example, with a specified XML file that contains imported data.

 **Note**

You may need to add the **DevExpress.Docs** reference to your application to access the **PdfViewer.Import** extension methods.

C#

```
(Form1.cs)
using System.Windows.Forms;
using DevExpress.Pdf;
namespace ImportAcroForm {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
            // Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\InitialAcroForm.pdf");
            // Import the document from an XML format.
            pdfViewer1.Import("..\\..\\FilledAcroForm.xml");
            // Save the imported document.
            pdfViewer1.SaveDocument("..\\..\\ImportedAcroForm.pdf");
        }
    }
}
```

Visual Basic

```
(Form1.vb)
Imports System.Windows.Forms
Imports DevExpress.Pdf
Namespace ImportAcroForm
    Partial Public Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
            ' Load a PDF document with AcroForm data.
            pdfViewer1.LoadDocument("..\\..\\InitialAcroForm.pdf")
            ' Import the document from an XML format.
            pdfViewer1.Import("..\\..\\FilledAcroForm.xml")
            ' Save the imported document.
            pdfViewer1.SaveDocument("..\\..\\ImportedAcroForm.pdf")
        End Sub
    End Class
End Namespace
```

See Also

[PdfViewerExtensions Class](#)

[PdfViewerExtensions Members](#)

[DevExpress.Pdf Namespace](#)

[PdfViewerExtensions.Import Overload List](#)

SaveDocument Method

Saves the document to the specified stream with encryption settings and document signature. This is an extension method.

Overload List

Name	Description
static bool SaveDocument(IPdfViewer viewer, Stream stream, PdfSaveOptions options)	Saves the document to the specified stream with encryption settings and document signature. This is an extension method.
static bool SaveDocument(IPdfViewer viewer, string path, PdfSaveOptions options)	Saves the document to the specified file path with encryption settings and document signature. This is an extension method.

See Also

[PdfViewerExtensions Class](#)
[PdfViewerExtensions Members](#)
[DevExpress.Pdf Namespace](#)
[PdfViewerExtensions.SaveDocument Overload List](#)

Saves the document to the specified stream with encryption settings and document signature. This is an extension method.

Namespace: [DevExpress.Pdf](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function SaveDocument(  
    ByVal viewer As IPdfViewer,  
    ByVal stream As Stream,  
    ByVal options As PdfSaveOptions  
) As Boolean
```

C#

```
public static bool SaveDocument(  
    IPdfViewer viewer,  
    Stream stream,  
    PdfSaveOptions options  
)
```

Parameters

viewer
A DevExpress.XtraPdfViewer.PdfViewer or DevExpress.Xpf.PdfViewer.PdfViewerControl object that implements the DevExpress.Pdf.IPdfViewer interface.

stream
A System.IO.Stream value specifying the location of the saved document.

options
A DevExpress.Pdf.PdfSaveOptions that contains the encryption and sign settings of a PDF document that should be saved.

Return Value

true, if the document is saved successfully; **false**, if the document saving operation is cancelled by the user.

Remarks

See the [Protecting a Document](#) and [Signing a Document](#) topics to learn more.

The **SaveDocument** method expects the input stream will not be closed or modified while a PDF document is saved (the DevExpress.XtraPdfViewer.PdfViewer.DetachStreamAfterLoadComplete or DevExpress.Xpf.PdfViewer.PdfViewerControl.DetachStreamOnLoadComplete property is set to **false** by default). Be aware that disposing of an output stream that has not been detached may cause errors on an attempt to apply further changes to a document. If you want to close the stream when a document is saved, set the DevExpress.XtraPdfViewer.PdfViewer.DetachStreamAfterLoadComplete (DevExpress.Xpf.PdfViewer.PdfViewerControl.DetachStreamOnLoadComplete) property to **true**.

See Also

[PdfViewerExtensions Class](#)
[PdfViewerExtensions Members](#)
[DevExpress.Pdf Namespace](#)
[PdfViewerExtensions.SaveDocument Overload List](#)

Saves the document to the specified file path with encryption settings and document signature. This is an extension method.

Namespace: [DevExpress.Pdf](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Function SaveDocument(  
    ByVal viewer As IPdfViewer,  
    ByVal path As String,  
    ByVal options As PdfSaveOptions  
) As Boolean
```

C#

```
public static bool SaveDocument(  
    IPdfViewer viewer,  
    string path,  
    PdfSaveOptions options  
)
```

Parameters

viewer

A DevExpress.XtraPdfViewer.PdfViewer or DevExpress.Xpf.PdfViewer.PdfViewerControl object that implements the DevExpress.Pdf.IPdfViewer interface.

path

A System.String, specifying the path to the directory to which the PDF document should be saved.

options

A DevExpress.Pdf.PdfSaveOptions that contains the encryption and sign settings of a PDF document that should be saved.

Return Value

true, if the document is saved successfully; **false**, if the document saving operation is cancelled by the user.

Remarks

See the [Protecting a Document](#) and [Signing a Document](#) topics to learn more.

Note

The PDF Viewer locks a file while a document is saved (the DevExpress.XtraPdfViewer.PdfViewer.DetachStreamAfterLoadComplete or DevExpress.Xpf.PdfViewer.PdfViewerControl.DetachStreamOnLoadComplete property is set to **false** by default). To unlock the file, set the DevExpress.XtraPdfViewer.PdfViewer.DetachStreamAfterLoadComplete (DevExpress.Xpf.PdfViewer.PdfViewerControl.DetachStreamOnLoadComplete) property to **true**.

See Also

[PdfViewerExtensions Class](#)
[PdfViewerExtensions Members](#)
[DevExpress.Pdf Namespace](#)
[PdfViewerExtensions.SaveDocument Overload List](#)

DevExpress.Snap

Contains the SnapControl class that provides the main functionality of Snap.

Classes

	Class	Description
	SnapDocumentServer	A non-visual reporting engine providing all the functionality of Snap.

SnapDocumentServer Class

A non-visual reporting engine providing all the functionality of Snap.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class SnapDocumentServer Inherits RichEditDocumentServer, ISnapDocumentServer, ISnapDriver
```

C#

```
public class SnapDocumentServer : RichEditDocumentServer, ISnapDocumentServer, ISnapDriver
```

Remarks

For a code sample, see [How to: Generate Master-Detail Mail Merge Documents](#).

Inheritance Hierarchy

[Object](#)

RichEditDocumentServer

SnapDocumentServer

See Also

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[Snap Report API](#)

[How to: Generate Master-Detail Mail Merge Documents](#)

SnapDocumentServer Members












A non-visual reporting engine providing all the functionality of Snap.

The following tables list the members exposed by the [SnapDocumentServer](#) type.

Public Constructors





	Name	Description
	SnapDocumentServer	Initializes a new instance of the SnapDocumentServer class with default settings.

Public Properties










	Name	Description
	Document	Provides access to a Snap document stored on the SnapDocumentServer .
	DocumentLayout	Provides access to the document layout. (Inherited from RichEditDocumentServer)
	DpiX	Gets the current dpi value for the X-coordinate. (Inherited from RichEditDocumentServer)
	DpiY	Gets the current dpi value for the Y-coordinate. (Inherited from RichEditDocumentServer)
	HtmlText	Gets or sets the content as HTML text. (Inherited from RichEditDocumentServer)
	IsPrintingAvailable	Maintained for compatibility with the code written for the RichEditControl. (Inherited from RichEditDocumentServer)
	IsUpdateLocked	Gets whether the object has been locked for updating. (Inherited from RichEditDocumentServer)
	LayoutCalculationMode	Gets or sets the mode for layout calculation. (Inherited from RichEditDocumentServer)
	LayoutUnit	Gets or sets a unit of measure used for a document model layout. (Inherited from RichEditDocumentServer)
	MhtText	Gets or sets the document content as MHT text. (Inherited from RichEditDocumentServer)
	Model	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)

	Modified	Gets or sets a value that indicates that the document contents is modified since it was last saved. (Inherited from RichEditDocumentServer)
	OpenDocumentBytes	Gets or sets the document content as an array of bytes in Open Office Text (.odt) format. (Inherited from RichEditDocumentServer)
	OpenXmlBytes	Gets or sets the document content as an array of bytes in Office Open XML (Docx) format. (Inherited from RichEditDocumentServer)
	Options	Provides access to the options that determine how a mail-merge document is displayed in a Snap application.
	RtfText	Gets or sets the formatted text content of the document. (Inherited from RichEditDocumentServer)
	SnxBytes	Specifies a byte array that stores a Snap document in the native SNX format.
	Text	Gets or sets the plain text content of the document. (Inherited from RichEditDocumentServer)
	Unit	Gets or sets a unit of measure used within the RichEditDocumentServer. (Inherited from RichEditDocumentServer)
	WordMLText	Gets or sets the document content as the text in WordProcessingML (Microsoft Office Word 2003 XML) format. (Inherited from RichEditDocumentServer)







Public Methods

	Name	Description
	AddService	Overloaded. Adds the specified service to the service container. (Inherited from RichEditDocumentServer)
	BeginInitialize	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	BeginUpdate	Locks the RichEditDocumentServer object by disallowing visual updates until the EndUpdate or CancelUpdate method is called. (Inherited from RichEditDocumentServer)
	CancelUpdate	Unlocks the RichEditDocumentServer object after it has been locked by the BeginUpdate method, without

		causing an immediate visual update. (Inherited from RichEditDocumentServer)
	CreateMailMergeOptions	Obsolete. Obsolete. Use the CreateSnapMailMergeExportOptions property instead.
	CreateNewDocument	Creates a new blank document. (Inherited from RichEditDocumentServer)
	CreateSnapMailMergeExportOptions	Creates the options that determine how a document is rendered when finishing a mail-merge report.
	Dispose	Releases resources associated with a RichEditDocumentServer instance. (Inherited from RichEditDocumentServer)
	EndUpdate	Unlocks the RichEditDocumentServer object after a call to the BeginUpdate method and causes an immediate visual update. (Inherited from RichEditDocumentServer)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	ExportDocument	Overloaded. Exports the document to a file in the specified format.
	ExportToPdf	Overloaded. Exports the document to the specified stream in PDF format. (Inherited from RichEditDocumentServer)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetService	Gets the service object of the specified type. (Inherited from RichEditDocumentServer)
	GetService<T>	Gets the specified service. (Inherited from RichEditDocumentServer)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	LoadDocument	Overloaded. Loads a document from the stream in the specified format. External content for HTML format is retrieved using the specified source (base) URI. (Inherited from RichEditDocumentServer)


	LoadDocumentTemplate	Overloaded. Loads a document from the stream. (Inherited from RichEditDocumentServer)
	MailMerge	Overloaded. Obsolete. Use the SnapMailMerge method instead.
	Print	Overloaded. Prints the document with the given printer settings. (Inherited from RichEditDocumentServer)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveService	Overloaded. Removes the service of the specified type from the service container. (Inherited from RichEditDocumentServer)
	ReplaceService<T>	Performs a service substitution. (Inherited from RichEditDocumentServer)
	SaveDocument	Overloaded. Saves the document to a stream in the Snap native document format (.SNX)
	SnapMailMerge	Overloaded. Starts rendering a mail-merge document and saving it to a stream in the specified format.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Events

	Name	Description
	AfterDataSourceImport	Occurs after importing a data source.
	AfterExport	Occurs after the document is successfully exported. (Inherited from RichEditDocumentServer)
	AsynchronousOperationFinished	Occurs after finishing an asynchronous operation in a separate thread.
	AsynchronousOperationStarted	Occurs after starting an asynchronous operation in a separate thread.
	BeforeConversion	Occurs before a snap document is exported to format other than the native .SNX.
	BeforeDataSourceExport	Occurs before exporting a data source.
	BeforeExport	Occurs before the document is saved (exported to a certain format). (Inherited from RichEditDocumentServer)
	BeforeImport	Occurs before a document is loaded (imported from an external source). (Inherited from RichEditDocumentServer)

	BeforeLoadCustomAssembly	Occurs when the a report template (.snx file) is loaded which contains the Entity Framework data source originated from a compiled assembly.
	BeforePagePaint	Enables you to specify a custom DevExpress.XtraRichEdit.API.Layout.PagePainter descendant to alter the way the layout elements are drawn. (Inherited from RichEditDocumentServer)
	CalculateDocumentVariable	Fires when the DOCVARIABLE field is updated. (Inherited from RichEditDocumentServer)
	CommentInserted	Occurs after a new comment is created in the document. (Inherited from RichEditDocumentServer)
	ContentChanged	Occurs when the document content was changed. (Inherited from RichEditDocumentServer)
	CustomAssemblyLoading	Obsolete. For internal use.
	CustomPropertiesChanged	Occurs when one of the DevExpress.XtraRichEdit.API.Native.DocumentCustomProperties has changed. (Inherited from RichEditDocumentServer)
	DataSourceChanged	Occurs when the data source has been changed.
	DocumentClosing	Occurs before closing a DevExpress.Snap.Core.API.SnapDocument.
	DocumentLoaded	Occurs before loading a DevExpress.Snap.Core.API.SnapDocument into a SnapDocumentServer .
	DocumentPropertiesChanged	Occurs after one of the DevExpress.XtraRichEdit.API.Native.DocumentProperties has changed. (Inherited from RichEditDocumentServer)
	EmptyDocumentCreated	Occurs before creating a new DevExpress.Snap.Core.API.SnapDocument.
	HtmlTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	InitializeDocument	Occurs before a document is loaded. Handle this event to set initial document settings. (Inherited from RichEditDocumentServer)
	InvalidFormatException	Fires when the supplied data could not be recognized as data in the assumed format for import. (Inherited from RichEditDocumentServer)

	MailMergeFinished	Obsolete. Obsolete. Use the SnapMailMergeFinished event instead.
	MailMergeRecordFinished	Obsolete. Obsolete. Use the SnapMailMergeRecordFinished event instead.
	MailMergeRecordStarted	Obsolete. Obsolete. Use the SnapMailMergeRecordStarted event instead.
	MailMergeStarted	Obsolete. Obsolete. Use the SnapMailMergeStarted event instead.
	MhtTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	ModifiedChanged	Occurs when the value of the DevExpress.XtraRichEdit.RichEditDocumentServer.Modified property is changed. (Inherited from RichEditDocumentServer)
	OpenDocumentBytesChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	OpenXmlBytesChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	RtfTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	SelectionChanged	Fires in response to changing a selection in the document. (Inherited from RichEditDocumentServer)
	SnapMailMergeFinished	Occurs after document merging has finished.
	SnapMailMergeRecordFinished	Occurs after data field merging has finished.
	SnapMailMergeRecordStarted	Occurs after data field merging has started.
	SnapMailMergeStarted	Occurs after document merging has started.
	UnhandledException	This event is raised when an exception unhandled by the RichEditDocumentServer occurs. (Inherited from RichEditDocumentServer)
	ValidateCustomSql	Allows validation of the custom SQL query created using the Data Source

		Wizard or the Query Builder.
	WordMLTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)

See Also

- [SnapDocumentServer Members](#)
- [DevExpress.Snap Namespace](#)
- [Snap Report API](#)
- [How to: Generate Master-Detail Mail Merge Documents](#)

SnapDocumentServer Constructor

Initializes a new instance of the [SnapDocumentServer](#) class with default settings.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public SnapDocumentServer()
```

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

SnapDocumentServer Properties












A non-visual reporting engine providing all the functionality of Snap.

The following tables list the members exposed by the [SnapDocumentServer](#) type.

Public Constructors





	Name	Description
	SnapDocumentServer	Initializes a new instance of the SnapDocumentServer class with default settings.

Public Properties










	Name	Description
	Document	Provides access to a Snap document stored on the SnapDocumentServer .
	DocumentLayout	Provides access to the document layout. (Inherited from RichEditDocumentServer)
	DpiX	Gets the current dpi value for the X-coordinate. (Inherited from RichEditDocumentServer)
	DpiY	Gets the current dpi value for the Y-coordinate. (Inherited from RichEditDocumentServer)
	HtmlText	Gets or sets the content as HTML text. (Inherited from RichEditDocumentServer)
	IsPrintingAvailable	Maintained for compatibility with the code written for the RichEditControl. (Inherited from RichEditDocumentServer)
	IsUpdateLocked	Gets whether the object has been locked for updating. (Inherited from RichEditDocumentServer)
	LayoutCalculationMode	Gets or sets the mode for layout calculation. (Inherited from RichEditDocumentServer)
	LayoutUnit	Gets or sets a unit of measure used for a document model layout. (Inherited from RichEditDocumentServer)
	MhtText	Gets or sets the document content as MHT text. (Inherited from RichEditDocumentServer)
	Model	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)

	Modified	Gets or sets a value that indicates that the document contents is modified since it was last saved. (Inherited from RichEditDocumentServer)
	OpenDocumentBytes	Gets or sets the document content as an array of bytes in Open Office Text (.odt) format. (Inherited from RichEditDocumentServer)
	OpenXmlBytes	Gets or sets the document content as an array of bytes in Office Open XML (Docx) format. (Inherited from RichEditDocumentServer)
	Options	Provides access to the options that determine how a mail-merge document is displayed in a Snap application.
	RtfText	Gets or sets the formatted text content of the document. (Inherited from RichEditDocumentServer)
	SnxBytes	Specifies a byte array that stores a Snap document in the native SNX format.
	Text	Gets or sets the plain text content of the document. (Inherited from RichEditDocumentServer)
	Unit	Gets or sets a unit of measure used within the RichEditDocumentServer. (Inherited from RichEditDocumentServer)
	WordMLText	Gets or sets the document content as the text in WordProcessingML (Microsoft Office Word 2003 XML) format. (Inherited from RichEditDocumentServer)









Public Methods

	Name	Description
	AddService	Overloaded. Adds the specified service to the service container. (Inherited from RichEditDocumentServer)
	BeginInitialize	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	BeginUpdate	Locks the RichEditDocumentServer object by disallowing visual updates until the EndUpdate or CancelUpdate method is called. (Inherited from RichEditDocumentServer)
	CancelUpdate	Unlocks the RichEditDocumentServer object after it has been locked by the BeginUpdate method, without

		causing an immediate visual update. (Inherited from RichEditDocumentServer)
	CreateMailMergeOptions	Obsolete. Obsolete. Use the CreateSnapMailMergeExportOptions property instead.
	CreateNewDocument	Creates a new blank document. (Inherited from RichEditDocumentServer)
	CreateSnapMailMergeExportOptions	Creates the options that determine how a document is rendered when finishing a mail-merge report.
	Dispose	Releases resources associated with a RichEditDocumentServer instance. (Inherited from RichEditDocumentServer)
	EndUpdate	Unlocks the RichEditDocumentServer object after a call to the BeginUpdate method and causes an immediate visual update. (Inherited from RichEditDocumentServer)
 	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	ExportDocument	Overloaded. Exports the document to a file in the specified format.
	ExportToPdf	Overloaded. Exports the document to the specified stream in PDF format. (Inherited from RichEditDocumentServer)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetService	Gets the service object of the specified type. (Inherited from RichEditDocumentServer)
	GetService<T>	Gets the specified service. (Inherited from RichEditDocumentServer)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	LoadDocument	Overloaded. Loads a document from the stream in the specified format. External content for HTML format is retrieved using the specified source (base) URI. (Inherited from RichEditDocumentServer)


	LoadDocumentTemplate	Overloaded. Loads a document from the stream. (Inherited from RichEditDocumentServer)
	MailMerge	Overloaded. Obsolete. Use the SnapMailMerge method instead.
	Print	Overloaded. Prints the document with the given printer settings. (Inherited from RichEditDocumentServer)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveService	Overloaded. Removes the service of the specified type from the service container. (Inherited from RichEditDocumentServer)
	ReplaceService<T>	Performs a service substitution. (Inherited from RichEditDocumentServer)
	SaveDocument	Overloaded. Saves the document to a stream in the Snap native document format (.SNX)
	SnapMailMerge	Overloaded. Starts rendering a mail-merge document and saving it to a stream in the specified format.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

Public Events

	Name	Description
	AfterDataSourceImport	Occurs after importing a data source.
	AfterExport	Occurs after the document is successfully exported. (Inherited from RichEditDocumentServer)
	AsynchronousOperationFinished	Occurs after finishing an asynchronous operation in a separate thread.
	AsynchronousOperationStarted	Occurs after starting an asynchronous operation in a separate thread.
	BeforeConversion	Occurs before a snap document is exported to format other than the native .SNX.
	BeforeDataSourceExport	Occurs before exporting a data source.
	BeforeExport	Occurs before the document is saved (exported to a certain format). (Inherited from RichEditDocumentServer)
	BeforeImport	Occurs before a document is loaded (imported from an external source). (Inherited from RichEditDocumentServer)

	BeforeLoadCustomAssembly	Occurs when the a report template (.snx file) is loaded which contains the Entity Framework data source originated from a compiled assembly.
	BeforePagePaint	Enables you to specify a custom DevExpress.XtraRichEdit.API.Layout.PagePainter descendant to alter the way the layout elements are drawn. (Inherited from RichEditDocumentServer)
	CalculateDocumentVariable	Fires when the DOCVARIABLE field is updated. (Inherited from RichEditDocumentServer)
	CommentInserted	Occurs after a new comment is created in the document. (Inherited from RichEditDocumentServer)
	ContentChanged	Occurs when the document content was changed. (Inherited from RichEditDocumentServer)
	CustomAssemblyLoading	Obsolete. For internal use.
	CustomPropertiesChanged	Occurs when one of the DevExpress.XtraRichEdit.API.Native.DocumentCustomProperties has changed. (Inherited from RichEditDocumentServer)
	DataSourceChanged	Occurs when the data source has been changed.
	DocumentClosing	Occurs before closing a DevExpress.Snap.Core.API.SnapDocument.
	DocumentLoaded	Occurs before loading a DevExpress.Snap.Core.API.SnapDocument into a SnapDocumentServer .
	DocumentPropertiesChanged	Occurs after one of the DevExpress.XtraRichEdit.API.Native.DocumentProperties has changed. (Inherited from RichEditDocumentServer)
	EmptyDocumentCreated	Occurs before creating a new DevExpress.Snap.Core.API.SnapDocument.
	HtmlTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	InitializeDocument	Occurs before a document is loaded. Handle this event to set initial document settings. (Inherited from RichEditDocumentServer)
	InvalidFormatException	Fires when the supplied data could not be recognized as data in the assumed format for import. (Inherited from RichEditDocumentServer)

	MailMergeFinished	Obsolete. Obsolete. Use the SnapMailMergeFinished event instead.
	MailMergeRecordFinished	Obsolete. Obsolete. Use the SnapMailMergeRecordFinished event instead.
	MailMergeRecordStarted	Obsolete. Obsolete. Use the SnapMailMergeRecordStarted event instead.
	MailMergeStarted	Obsolete. Obsolete. Use the SnapMailMergeStarted event instead.
	MhtTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	ModifiedChanged	Occurs when the value of the DevExpress.XtraRichEdit.RichEditDocumentServer.Modified property is changed. (Inherited from RichEditDocumentServer)
	OpenDocumentBytesChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	OpenXmlBytesChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	RtfTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	SelectionChanged	Fires in response to changing a selection in the document. (Inherited from RichEditDocumentServer)
	SnapMailMergeFinished	Occurs after document merging has finished.
	SnapMailMergeRecordFinished	Occurs after data field merging has finished.
	SnapMailMergeRecordStarted	Occurs after data field merging has started.
	SnapMailMergeStarted	Occurs after document merging has started.
	UnhandledException	This event is raised when an exception unhandled by the RichEditDocumentServer occurs. (Inherited from RichEditDocumentServer)
	ValidateCustomSql	Allows validation of the custom SQL query created using the Data Source

		Wizard or the Query Builder.
	WordMLTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)

See Also
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[Snap Report API](#)
[How to: Generate Master-Detail Mail Merge Documents](#)

Document Property

Provides access to a Snap document stored on the [SnapDocumentServer](#).

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Document As SnapDocument
```

C#

```
public SnapDocument Document { get; }
```

Property Value

A DevExpress.Snap.Core.API.SnapDocument object.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

Options Property

Provides access to the options that determine how a [mail-merge document](#) is displayed in a Snap application.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Options As SnapDocumentServerOptions
```

C#

```
public SnapDocumentServerOptions Options { get; }
```

Property Value

A SnapDocumentServerOptions object.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

DevExpress.Snap.SnapControlOptions.SnapMailMergeVisualOptions

SnxBytes Property

Specifies a byte array that stores a Snap document in the native SNX format.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property SnxBytes As Byte[]
```

C#

```
public Byte[] SnxBytes { get; set; }
```

Property Value

A [System.Byte](#) array.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

SnapDocumentServer Events

A non-visual reporting engine providing all the functionality of Snap.

The following tables list the members exposed by the [SnapDocumentServer](#) type.

Public Events

	Name	Description
	AfterDataSourceImport	Occurs after importing a data source.
	AfterExport	Occurs after the document is successfully exported. (Inherited from RichEditDocumentServer)
	AsynchronousOperationFinished	Occurs after finishing an asynchronous operation in a separate thread.
	AsynchronousOperationStarted	Occurs after starting an asynchronous operation in a separate thread.
	BeforeConversion	Occurs before a snap document is exported to format other than the native .SNX.
	BeforeDataSourceExport	Occurs before exporting a data source.
	BeforeExport	Occurs before the document is saved (exported to a certain format). (Inherited from RichEditDocumentServer)
	BeforeImport	Occurs before a document is loaded (imported from an external source). (Inherited from RichEditDocumentServer)
	BeforeLoadCustomAssembly	Occurs when the a report template (.snx file) is loaded which contains the Entity Framework data source originated from a compiled assembly.
	BeforePagePaint	Enables you to specify a custom DevExpress.XtraRichEdit.API.Layout.Painter descendant to alter the way the layout elements are drawn. (Inherited from RichEditDocumentServer)
	CalculateDocumentVariable	Fires when the DOCVARIABLE field is updated. (Inherited from RichEditDocumentServer)
	CommentInserted	Occurs after a new comment is created in the document. (Inherited from RichEditDocumentServer)
	ContentChanged	Occurs when the document content was changed. (Inherited from RichEditDocumentServer)
	CustomAssemblyLoading	Obsolete. For internal use.
	CustomPropertiesChanged	Occurs when one of the DevExpress.XtraRichEdit.API.Native.DocumentCustomProperties has changed.

		(Inherited from RichEditDocumentServer)
	DataSourceChanged	Occurs when the data source has been changed.
	DocumentClosing	Occurs before closing a DevExpress.Snap.Core.API.SnapDocument.
	DocumentLoaded	Occurs before loading a DevExpress.Snap.Core.API.SnapDocument into a SnapDocumentServer .
	DocumentPropertiesChanged	Occurs after one of the DevExpress.XtraRichEdit.API.Native.DocumentProperties has changed. (Inherited from RichEditDocumentServer)
	EmptyDocumentCreated	Occurs before creating a new DevExpress.Snap.Core.API.SnapDocument.
	HtmlTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	InitializeDocument	Occurs before a document is loaded. Handle this event to set initial document settings. (Inherited from RichEditDocumentServer)
	InvalidFormatException	Fires when the supplied data could not be recognized as data in the assumed format for import. (Inherited from RichEditDocumentServer)
	MailMergeFinished	Obsolete. Obsolete. Use the SnapMailMergeFinished event instead.
	MailMergeRecordFinished	Obsolete. Obsolete. Use the SnapMailMergeRecordFinished event instead.
	MailMergeRecordStarted	Obsolete. Obsolete. Use the SnapMailMergeRecordStarted event instead.
	MailMergeStarted	Obsolete. Obsolete. Use the SnapMailMergeStarted event instead.
	MhtTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	ModifiedChanged	Occurs when the value of the DevExpress.XtraRichEdit.RichEditDocumentServer.Modified property is changed. (Inherited from RichEditDocumentServer)
	OpenDocumentBytesChanged	This member supports the internal infrastructure and is not intended to be

		used directly from your code. (Inherited from RichEditDocumentServer)
	OpenXmlBytesChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	RtfTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	SelectionChanged	Fires in response to changing a selection in the document. (Inherited from RichEditDocumentServer)
	SnapMailMergeFinished	Occurs after document merging has finished.
	SnapMailMergeRecordFinished	Occurs after data field merging has finished.
	SnapMailMergeRecordStarted	Occurs after data field merging has started.
	SnapMailMergeStarted	Occurs after document merging has started.
	UnhandledException	This event is raised when an exception unhandled by the RichEditDocumentServer occurs. (Inherited from RichEditDocumentServer)
	ValidateCustomSql	Allows validation of the custom SQL query created using the Data Source Wizard or the Query Builder.
	WordMLTextChanged	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)

See Also[SnapDocumentServer Members](#)[DevExpress.Snap Namespace](#)[Snap Report API](#)[How to: Generate Master-Detail Mail Merge Documents](#)

AfterDataSourceImport Event

Occurs after importing a data source.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public AfterDataSourceImport As AfterDataSourceImportEventHandler
```

C#

```
public event AfterDataSourceImportEventHandler AfterDataSourceImport
```

Event Data

The event handler receives an argument of type `AfterDataSourceImportEventArgs` containing data related to this event.

Remarks

Handle the **AfterDataSourceImport** event to load the custom data that has been saved using the [BeforeDataSourceExport](#) event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[BeforeDataSourceExport](#)

AsynchronousOperationFinished Event

Occurs after finishing an asynchronous operation in a separate thread.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
event Public AsynchronousOperationFinished As [EventHandler](#)
C#
public event [EventHandler](#) AsynchronousOperationFinished

Event Data

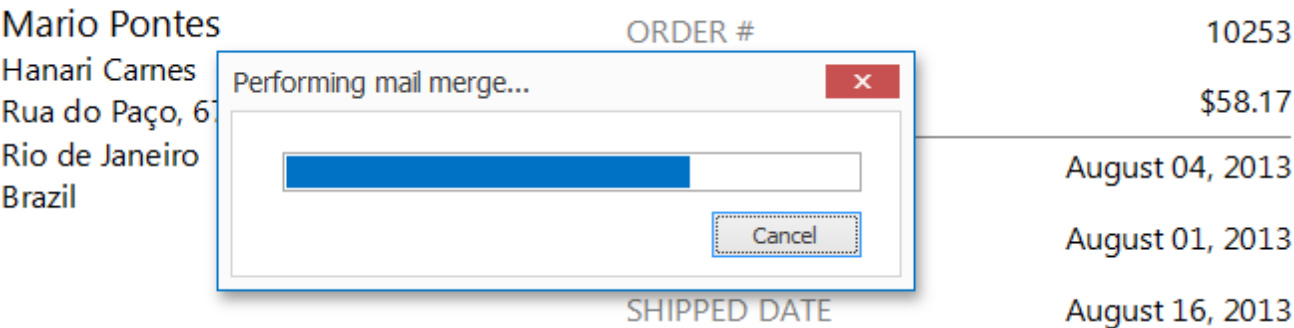
The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

Handle this event to track the status of asynchronous operations that are being currently performed in separate threads and avoid simultaneous execution of similar operations.

The document assembly is processed to an entirely separate thread to avoid application lock-ups.

An example of an asynchronous operation that is performed in a separate thread is the **Finish & Merge** command that triggers the assembling of a mail merge document.



See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[AsynchronousOperationStarted](#)

AsynchronousOperationStarted Event

Occurs after starting an asynchronous operation in a separate thread.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
event Public AsynchronousOperationStarted As [EventHandler](#)
C#
public event [EventHandler](#) AsynchronousOperationStarted

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

Handle this event to track the status of asynchronous operations that are being currently performed in separate threads and avoid simultaneous execution of similar operations.

The document assembly is processed to an entirely separate thread to avoid application lock-ups.

An example of an asynchronous operation that is performed in a separate thread is the **Finish & Merge** command that triggers the assembling of a mail merge document.

Mario Pontes
Hanari Carnes
Rua do Paço, 67
Rio de Janeiro
Brazil

ORDER #
10253
\$58.17
August 04, 2013
August 01, 2013
August 16, 2013

Performing mail merge...

Cancel

SHIPPED DATE

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[AsynchronousOperationFinished](#)

BeforeConversion Event

Occurs before a snap document is exported to format other than the native .SNX.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public BeforeConversion As BeforeConversionEventHandler
```

C#

```
public event BeforeConversionEventHandler BeforeConversion
```

Event Data

The event handler receives an argument of type `BeforeConversionEventArgs` containing data related to this event.

Remarks

The **BeforeConversion** allows you to implement a custom export procedure.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

BeforeDataSourceExport Event

Occurs before exporting a data source.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public BeforeDataSourceExport As BeforeDataSourceExportEventHandler
```

C#

```
public event BeforeDataSourceExportEventHandler BeforeDataSourceExport
```

Event Data

The event handler receives an argument of type `BeforeDataSourceExportEventArgs` containing data related to this event.

Remarks

Handle the **BeforeDataSourceExport** event to provide a custom mechanism for saving a data source along with the document, by assigning a [System.Byte](#) array to the `DevExpress.Snap.BeforeDataSourceExportEventArgs.Data` property.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[AfterDataSourceImport](#)

BeforeLoadCustomAssembly Event

Occurs when the a report template (.snx file) is loaded which contains the Entity Framework data source originated from a compiled assembly.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public BeforeLoadCustomAssembly As BeforeLoadCustomAssemblyEventHandler
```

C#

```
public event BeforeLoadCustomAssemblyEventHandler BeforeLoadCustomAssembly
```

Event Data

The event handler receives an argument of type BeforeLoadCustomAssemblyEventArgs containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

CustomAssemblyLoading Event

NOTE: This type is now obsolete.

This event is not appropriate in Snap and has been rendered obsolete. For details, refer to BC3900

For internal use.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public CustomAssemblyLoading As EventHandler(of CustomAssemblyLoadingEventArgs)
```

C#

```
public event EventHandler<CustomAssemblyLoadingEventArgs> CustomAssemblyLoading
```

Event Data

The event handler receives an argument of type CustomAssemblyLoadingEventArgs containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

DataSourceChanged Event

Occurs when the data source has been changed.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public DataSourceChanged As EventHandler
```

C#

```
public event EventHandler DataSourceChanged
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

Handle the **DataSourceChanged** event to perform actions when the report's data source changes.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

DocumentClosing Event

Occurs before closing a DevExpress.Snap.Core.API.SnapDocument.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public DocumentClosing As DocumentClosingEventHandler
```

C#

```
public event DocumentClosingEventHandler DocumentClosing
```

Event Data

The event handler receives an argument of type DocumentClosingEventArgs containing data related to this event.

Remarks

Handle this event to obtain interim document data sources that have been created by an end-user (e.g., to save them for another document editing session) via the DevExpress.Snap.DocumentClosingEventArgs.InterimDataSources property.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[DocumentLoaded](#)

[EmptyDocumentCreated](#)

DocumentLoaded Event

Occurs before loading a DevExpress.Snap.Core.API.SnapDocument into a [SnapDocumentServer](#).

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public DocumentLoaded As DocumentImportedEventHandler
```

C#

```
public event DocumentImportedEventHandler DocumentLoaded
```

Event Data

The event handler receives an argument of type DocumentImportedEventArgs containing data related to this event.

Remarks

Handle this event to obtain interim document data sources that have been created by an end-user (e.g., to save them for another document editing session) via the DevExpress.Snap.DocumentImportedEventArgs.InterimDataSources property.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[EmptyDocumentCreated](#)

[DocumentClosing](#)

EmptyDocumentCreated Event

Occurs before creating a new DevExpress.Snap.Core.API.SnapDocument.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public EmptyDocumentCreated As DocumentImportedEventHandler
```

C#

```
public event DocumentImportedEventHandler EmptyDocumentCreated
```

Event Data

The event handler receives an argument of type DocumentImportedEventArgs containing data related to this event.
The following **DocumentImportedEventArgs** properties provide information specific to this event.

Property	Description
Handled	Specifies whether or not the corresponding event of the DevExpress.Snap.SnapControl was handled.
InterimDataSources	Provides access to the interim document data sources that have been created by an end-user during a document editing session.

Remarks

Handle this event to obtain interim document data sources that have been created by an end-user (e.g., to save them for another document editing session) via the DevExpress.Snap.DocumentImportedEventArgs.InterimDataSources property.

See Also

- [SnapDocumentServer Class](#)
- [SnapDocumentServer Members](#)
- [DevExpress.Snap Namespace](#)
- [DocumentLoaded](#)
- [DocumentClosing](#)

MailMergeFinished Event

NOTE: This type is now obsolete.

This event is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMergeFinished](#) event instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public MailMergeFinished As MailMergeFinishedEventHandler
```

C#

```
public event MailMergeFinishedEventHandler MailMergeFinished
```

Event Data

The event handler receives an argument of type `MailMergeFinishedEventArgs` containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapMailMergeFinished](#)

MailMergeRecordFinished Event

NOTE: This type is now obsolete.

This event is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMergeRecordFinished](#) event instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public MailMergeRecordFinished As MailMergeRecordFinishedEventHandler
```

C#

```
public event MailMergeRecordFinishedEventHandler MailMergeRecordFinished
```

Event Data

The event handler receives an argument of type `MailMergeRecordFinishedEventArgs` containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapMailMergeRecordFinished](#)

MailMergeRecordStarted Event

NOTE: This type is now obsolete.

This event is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMergeRecordStarted](#) event instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public MailMergeRecordStarted As MailMergeRecordStartedEventHandler
```

C#

```
public event MailMergeRecordStartedEventHandler MailMergeRecordStarted
```

Event Data

The event handler receives an argument of type `MailMergeRecordStartedEventArgs` containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapMailMergeRecordStarted](#)

MailMergeStarted Event

NOTE: This type is now obsolete.

This event is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMergeStarted](#) event instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public MailMergeStarted As MailMergeStartedEventHandler
```

C#

```
public event MailMergeStartedEventHandler MailMergeStarted
```

Event Data

The event handler receives an argument of type `MailMergeStartedEventArgs` containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapMailMergeStarted](#)

SnapMailMergeFinished Event

Occurs after document merging has finished.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SnapMailMergeFinished As SnapMailMergeFinishedEventHandler
```

C#

```
public event SnapMailMergeFinishedEventHandler SnapMailMergeFinished
```

Event Data

The event handler receives an argument of type `SnapMailMergeFinishedEventArgs` containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

SnapMailMergeRecordFinished Event

Occurs after data field merging has finished.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SnapMailMergeRecordFinished As SnapMailMergeRecordFinishedEventHandler
```

C#

```
public event SnapMailMergeRecordFinishedEventHandler SnapMailMergeRecordFinished
```

Event Data

The event handler receives an argument of type SnapMailMergeRecordFinishedEventArgs containing data related to this event.

Remarks

Handle the **SnapMailMergeRecordFinished** event to perform any actions after the process of field merging has finished. For example, you can access the data source collection of a single record document using the DevExpress.Snap.Core.API.SnapDocument.DataSources property of the DevExpress.Snap.Core.API.SnapDocument object accessed via the DevExpress.Snap.SnapMailMergeRecordFinishedEventArgs.RecordDocument property.

Note

Do not delete or modify the main data source used for the mail-merge process (i.e., the data source assigned to the **DataSource** and **DataSourceName** properties of the DevExpress.Snap.Core.Options.SnapMailMergeVisualOptions object) in the event handler, an exception will be thrown otherwise.

See Also

[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)

SnapMailMergeRecordStarted Event

Occurs after data field merging has started.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SnapMailMergeRecordStarted As SnapMailMergeRecordStartedEventHandler
```

C#

```
public event SnapMailMergeRecordStartedEventHandler SnapMailMergeRecordStarted
```

Event Data

The event handler receives an argument of type `SnapMailMergeRecordStartedEventArgs` containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

SnapMailMergeStarted Event

Occurs after document merging has started.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SnapMailMergeStarted As SnapMailMergeStartedEventHandler
```

C#

```
public event SnapMailMergeStartedEventHandler SnapMailMergeStarted
```

Event Data

The event handler receives an argument of type `SnapMailMergeStartedEventArgs` containing data related to this event.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

ValidateCustomSql Event

Allows validation of the custom SQL query created using the Data Source Wizard or the Query Builder.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ValidateCustomSql As EventHandler(of ValidateSqlEventArgs)
```

C#

```
public event EventHandler<ValidateSqlEventArgs> ValidateCustomSql
```

Event Data

The event handler receives an argument of type `ValidateSqlEventArgs` containing data related to this event.

Remarks

The **ValidateCustomSql** event occurs when the end-user enters custom SQL query text using the Data Source Wizard or the Query Builder, if the **SnapControl.Options.DataSourceWizardOptions.SqlWizardSettings.EnableCustomSql** property is set to **true**.

The `DevExpress.DataAccess.ValidateCustomSqlQueryEventArgs.CustomSqlQuery` property returns the text of SQL query. You can determine whether the text is valid, using custom criteria, and set the `DevExpress.DataAccess.ValidateCustomSqlQueryEventArgs.Valid` to **true** to proceed with the text or **false** to cancel and display a message specified by the `DevExpress.DataAccess.ValidateCustomSqlQueryEventArgs.ExceptionMessage` property.



Note

If the **ValidateCustomSql** event is not handled, a custom query used to obtain data from the SQL database should contain only SELECT statements (default validation criterion).

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

SnapDocumentServer Methods

A non-visual reporting engine providing all the functionality of Snap.

The following tables list the members exposed by the [SnapDocumentServer](#) type.

Public Methods

	Name	Description
	AddService	Overloaded. Adds the specified service to the service container. (Inherited from RichEditDocumentServer)
	BeginInitialize	This member supports the internal infrastructure and is not intended to be used directly from your code. (Inherited from RichEditDocumentServer)
	BeginUpdate	Locks the RichEditDocumentServer object by disallowing visual updates until the EndUpdate or CancelUpdate method is called. (Inherited from RichEditDocumentServer)
	CancelUpdate	Unlocks the RichEditDocumentServer object after it has been locked by the BeginUpdate method, without causing an immediate visual update. (Inherited from RichEditDocumentServer)
	CreateMailMergeOptions	Obsolete. Obsolete. Use the CreateSnapMailMergeExportOptions property instead.
	CreateNewDocument	Creates a new blank document. (Inherited from RichEditDocumentServer)
	CreateSnapMailMergeExportOptions	Creates the options that determine how a document is rendered when finishing a mail-merge report.
	Dispose	Releases resources associated with a RichEditDocumentServer instance. (Inherited from RichEditDocumentServer)
	EndUpdate	Unlocks the RichEditDocumentServer object after a call to the BeginUpdate method and causes an immediate visual update. (Inherited from RichEditDocumentServer)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)

	ExportDocument	Overloaded. Exports the document to a file in the specified format.
	ExportToPdf	Overloaded. Exports the document to the specified stream in PDF format. (Inherited from RichEditDocumentServer)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetService	Gets the service object of the specified type. (Inherited from RichEditDocumentServer)
	GetService<T>	Gets the specified service. (Inherited from RichEditDocumentServer)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	LoadDocument	Overloaded. Loads a document from the stream in the specified format. External content for HTML format is retrieved using the specified source (base) URI. (Inherited from RichEditDocumentServer)
	LoadDocumentTemplate	Overloaded. Loads a document from the stream. (Inherited from RichEditDocumentServer)
	MailMerge	Overloaded. Obsolete. Use the SnapMailMerge method instead.
	Print	Overloaded. Prints the document with the given printer settings. (Inherited from RichEditDocumentServer)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveService	Overloaded. Removes the service of the specified type from the service container. (Inherited from RichEditDocumentServer)
	ReplaceService<T>	Performs a service substitution. (Inherited from RichEditDocumentServer)
	SaveDocument	Overloaded. Saves the document to a stream in the Snap native document format (.SNX)
	SnapMailMerge	Overloaded. Starts rendering a mail-merge document and saving it to a stream in the specified format.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also[SnapDocumentServer Members](#)[DevExpress.Snap Namespace](#)[Snap Report API](#)[How to: Generate Master-Detail Mail Merge Documents](#)

CreateMailMergeOptions Method

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [CreateSnapMailMergeExportOptions](#) property instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CreateMailMergeOptions() As MailMergeOptions
```

C#

```
public MailMergeOptions CreateMailMergeOptions()
```

Return Value

An object implementing the DevExpress.XtraRichEdit.API.Native.MailMergeOptions interface.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[CreateSnapMailMergeExportOptions](#)

CreateSnapMailMergeExportOptions Method

Creates the options that determine how a document is rendered when finishing a mail-merge report.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CreateSnapMailMergeExportOptions() As SnapMailMergeExportOptions
```

C#

```
public SnapMailMergeExportOptions CreateSnapMailMergeExportOptions()
```

Return Value

A DevExpress.Snap.Core.Options.SnapMailMergeExportOptions object.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

ExportDocument Method

Exports the document to a file in the specified format.

Overload List

Name	Description
void ExportDocument(string fileName, DocumentFormat documentFormat)	Exports the document to a file in the specified format.
void ExportDocument(Stream stream, DocumentFormat documentFormat)	Exports the document to a stream in the specified format.

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.ExportDocument Overload List](#)

Exports the document to a file in the specified format.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportDocument(  
    ByVal fileName As String,  
    ByVal documentFormat As DocumentFormat  
)
```

C#

```
public void ExportDocument(  
    string fileName,  
    DocumentFormat documentFormat  
)
```

Parameters

fileName
A string value containing the full path (including the file name) specifying where the document will be saved.
documentFormat
A DevExpress.XtraRichEdit.DocumentFormat structure that specifies the format of the exported document.

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.ExportDocument Overload List](#)
[SaveDocument](#)

Exports the document to a stream in the specified format.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportDocument(  
    ByVal stream As Stream,  
    ByVal documentFormat As DocumentFormat  
)
```

C#

```
public void ExportDocument(  
    Stream stream,  
    DocumentFormat documentFormat  
)
```

Parameters

stream

A [System.IO.Stream](#) object to output the document to.

documentFormat

A DevExpress.XtraRichEdit.DocumentFormat structure that specifies the format of the exported document.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.ExportDocument Overload List](#)

[SaveDocument](#)

MailMerge Method

Obsolete. Use the [SnapMailMerge](#) method instead.

Overload List

Name	Description
void MailMerge(Document document)	Obsolete. Use the SnapMailMerge method instead.
void MailMerge(IRichEditDocumentServer documentServer)	Obsolete. Use the SnapMailMerge method instead.
void MailMerge(MailMergeOptions options, Document targetDocument)	Obsolete. Use the SnapMailMerge method instead.
void MailMerge(MailMergeOptions options, IRichEditDocumentServer targetDocumentServer)	Obsolete. Use the SnapMailMerge method instead.
void MailMerge(Stream stream, DocumentFormat format)	Obsolete. Use the SnapMailMerge method instead.
void MailMerge(string fileName, DocumentFormat format)	Obsolete. Use the SnapMailMerge method instead.
void MailMerge(MailMergeOptions options, Stream stream, DocumentFormat format)	Obsolete. Use the SnapMailMerge method instead.
void MailMerge(MailMergeOptions options, string fileName, DocumentFormat format)	Obsolete. Use the SnapMailMerge method instead.

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.MailMerge Overload List](#)

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMerge](#) method instead.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub MailMerge(  
    ByVal document As Document  
)
```

C#

```
public void MailMerge(  
    Document document  
)
```

Parameters

document
An object implementing the DevExpress.XtraRichEdit.API.Native.Document interface.

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.MailMerge Overload List](#)
Master-Detail Mail Merge
[Snap Report API](#)
[SnapMailMerge](#)

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMerge](#) method instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub MailMerge(  
    ByVal documentServer As IRichEditDocumentServer  
)
```

C#

```
public void MailMerge(  
    IRichEditDocumentServer documentServer  
)
```

Parameters

documentServer

An object implementing the DevExpress.XtraRichEdit.IRichEditDocumentServer interface.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.MailMerge Overload List](#)

Master-Detail Mail Merge

[Snap Report API](#)

[SnapMailMerge](#)

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMerge](#) method instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub MailMerge(  
    ByVal options As MailMergeOptions,  
    ByVal targetDocument As Document  
)
```

C#

```
public void MailMerge(  
    MailMergeOptions options,  
    Document targetDocument  
)
```

Parameters

options

An object implementing the DevExpress.XtraRichEdit.API.Native.MailMergeOptions interface.

targetDocument

An object implementing the DevExpress.XtraRichEdit.API.Native.Document interface.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.MailMerge Overload List](#)

Master-Detail Mail Merge

[Snap Report API](#)

[SnapMailMerge](#)

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMerge](#) method instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub MailMerge(  
    ByVal options As MailMergeOptions,  
    ByVal targetDocumentServer As IRichEditDocumentServer  
)
```

C#

```
public void MailMerge(  
    MailMergeOptions options,  
    IRichEditDocumentServer targetDocumentServer  
)
```

Parameters

options

An object implementing the DevExpress.XtraRichEdit.API.Native.MailMergeOptions interface.

targetDocumentServer

An object implementing the DevExpress.XtraRichEdit.IRichEditDocumentServer interface.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.MailMerge Overload List](#)

Master-Detail Mail Merge

[Snap Report API](#)

[SnapMailMerge](#)

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMerge](#) method instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub MailMerge(  
    ByVal stream As Stream,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void MailMerge(  
    Stream stream,  
    DocumentFormat format  
)
```

Parameters

stream

A [System.IO.Stream](#) object.

format

A DevExpress.XtraRichEdit.DocumentFormat structure.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.MailMerge Overload List](#)

Master-Detail Mail Merge

[Snap Report API](#)

[SnapMailMerge](#)

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMerge](#) method instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub MailMerge(  
    ByVal fileName As String,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void MailMerge(  
    string fileName,  
    DocumentFormat format  
)
```

Parameters

fileName

A [System.String](#) value.

format

A DevExpress.XtraRichEdit.DocumentFormat structure.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.MailMerge Overload List](#)

Master-Detail Mail Merge

[Snap Report API](#)

[SnapMailMerge](#)

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMerge](#) method instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub MailMerge(  
    ByVal options As MailMergeOptions,  
    ByVal stream As Stream,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void MailMerge(  
    MailMergeOptions options,  
    Stream stream,  
    DocumentFormat format  
)
```

Parameters

options

An object implementing the DevExpress.XtraRichEdit.API.Native.MailMergeOptions interface.

stream

A [System.IO.Stream](#) object.

format

A DevExpress.XtraRichEdit.DocumentFormat structure.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.MailMerge Overload List](#)

Master-Detail Mail Merge

[Snap Report API](#)

[SnapMailMerge](#)

NOTE: This member is now obsolete.

This method is not appropriate in Snap and has been rendered obsolete.

Obsolete. Use the [SnapMailMerge](#) method instead.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub MailMerge(  
    ByVal options As MailMergeOptions,  
    ByVal fileName As String,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void MailMerge(  
    MailMergeOptions options,  
    string fileName,  
    DocumentFormat format  
)
```

Parameters

options

An object implementing the DevExpress.XtraRichEdit.API.Native.MailMergeOptions interface.

fileName

A [System.String](#) value.

format

A DevExpress.XtraRichEdit.DocumentFormat structure.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.MailMerge Overload List](#)

Master-Detail Mail Merge

[Snap Report API](#)

[SnapMailMerge](#)

SaveDocument Method

Saves the document to a file in the Snap native document format (.SNX)

Overload List

Name	Description
void SaveDocument(string fileName)	Saves the document to a file in the Snap native document format (.SNX)
void SaveDocument(Stream stream)	Saves the document to a stream in the Snap native document format (.SNX)
void SaveDocument(string fileName, DocumentFormat documentFormat)	Saves the document to a file in the specified format.
void SaveDocument(Stream stream, DocumentFormat documentFormat)	Saves the document to a stream in the specified format.

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.SaveDocument Overload List](#)

Saves the document to a file in the Snap native document format (.SNX)

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Overridable Public Sub SaveDocument(  
    ByVal fileName As String  
)
```

C#

```
public virtual void SaveDocument(  
    string fileName  
)
```

Parameters

fileName
A string value specifying the path to a file in which to save the document.

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.SaveDocument Overload List](#)
[ExportDocument](#)

Saves the document to a stream in the Snap native document format (.SNX)

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Overridable Public Sub SaveDocument(  
    ByVal stream As Stream  
)
```

C#

```
public virtual void SaveDocument(  
    Stream stream  
)
```

Parameters

stream

The [System.IO.Stream](#) object to output the document to.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.SaveDocument Overload List](#)

[ExportDocument](#)

NOTE: This member is now obsolete.

This method overload has become obsolete. Use the Save method with the "fileName" parameter to save your documents in the native Snap format (.snx), and the Export method - to store your documents in other formats.

Saves the document to a file in the specified format.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub SaveDocument(  
    ByVal fileName As String,  
    ByVal documentFormat As DocumentFormat  
)
```

C#

```
public void SaveDocument(  
    string fileName,  
    DocumentFormat documentFormat  
)
```

Parameters

fileName

A [System.String](#) value specifying the file name.

documentFormat

A DevExpress.XtraRichEdit.DocumentFormat structure.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.SaveDocument Overload List](#)

NOTE: This member is now obsolete.

This method overload has become obsolete. Use the Save method with the "fileName" parameter to save your documents in the native Snap format (.snx), and the Export method - to store your documents in other formats.

Saves the document to a stream in the specified format.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub SaveDocument(  
    ByVal stream As Stream,  
    ByVal documentFormat As DocumentFormat  
)
```

C#

```
public void SaveDocument(  
    Stream stream,  
    DocumentFormat documentFormat  
)
```

Parameters

stream

A [System.IO.Stream](#) object.

documentFormat

A DevExpress.XtraRichEdit.DocumentFormat structure.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.SaveDocument Overload List](#)

SnapMailMerge Method

Obsolete. Starts rendering a mail-merge document and saving it to the specified [Snap Report API](#).

Overload List

Name	Description
void SnapMailMerge(ISnapDocumentServer documentServer)	Obsolete. Starts rendering a mail-merge document and saving it to the specified Snap Report API .
void SnapMailMerge(SnapDocument document)	Starts rendering the specified mail-merge document.
void SnapMailMerge(Stream stream, DocumentFormat format)	Starts rendering a mail-merge document and saving it to a stream in the specified format.
void SnapMailMerge(SnapMailMergeExportOptions options, ISnapDocumentServer targetDocumentServer)	Obsolete. Starts rendering a mail-merge document based on the applied export options and saving it to the specified Snap Report API .
void SnapMailMerge(SnapMailMergeExportOptions options, SnapDocument targetDocument)	Starts rendering a mail-merge document based on the applied export options and saving it to the specified target document.
void SnapMailMerge(string fileName, DocumentFormat format)	Starts rendering a mail-merge document and saving it to a file in the specified format.
void SnapMailMerge(SnapMailMergeExportOptions options, string fileName, DocumentFormat format)	Starts rendering a mail-merge document based on the applied export options and saving it to a file in the specified format.
void SnapMailMerge(SnapMailMergeExportOptions options, Stream stream, DocumentFormat format)	Starts rendering a mail-merge document based on the applied export options and saving it to a stream in the specified format.

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.SnapMailMerge Overload List](#)

NOTE: This member is now obsolete.

This method has become obsolete. Use another SnapMailMerge method overload with appropriate parameters instead.

Obsolete. Starts rendering a mail-merge document and saving it to the specified [Snap Report API](#).

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SnapMailMerge(  
    ByVal documentServer As ISnapDocumentServer  
)
```

C#

```
public void SnapMailMerge(  
    ISnapDocumentServer documentServer  
)
```

Parameters

documentServer
A [SnapDocumentServer](#) object.

See Also

[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.SnapMailMerge Overload List](#)
[Snap Report API](#)
Developer Guidelines
Snap Mail Merge

Starts rendering the specified mail-merge document.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SnapMailMerge(  
    ByVal document As SnapDocument  
)
```

C#

```
public void SnapMailMerge(  
    SnapDocument document  
)
```

Parameters

document
An object implementing the DevExpress.Snap.Core.API.SnapDocument interface.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E5078>.

The following code uses the Snap Report API to generate a mail-merge report in the context of the [DevExpress Office File API](#).

It is a console application that creates a document and then exports it to an RTF file.

To use the [SnapDocumentServer](#), the project requires references to the following assemblies:

- DevExpress.Data.v18.1.dll
- DevExpress.Docs.v18.1.dll
- DevExpress.Office.v18.1.Core.dll
- DevExpress.RichEdit.v18.1.Core.dll
- DevExpress.Snap.v18.1.Core.dll

To perform mail merge, the application requires a template - a file in the native .SNX format which contains the document layout, specific fields and data source information. The template file must be created beforehand using the Snap Design Surface. For an example of template creation, review the Lesson 3 - Create a Mail Merge Report.

The code handles the [SnapMailMergeRecordStarted](#) and [SnapMailMergeRecordFinished](#) events to make adjustments to the merged document created for the particular data record.

The server loads a mail merge template using the DevExpress.XtraRichEdit.RichEditDocumentServer.LoadDocument method. Subsequently, the code assigns the data source by specifying the DevExpress.Snap.Core.API.IDataSourceOwner.DataSource property. The mail merge operation is performed by calling the [SnapMailMerge](#) method.

C#

```
(Program.cs)
using DevExpress.Snap;
using DevExpress.Snap.Core.API;
using DevExpress.XtraRichEdit;
using MailMergeServer.nwindDataSetTableAdapters;
using System;
using System.Data.OleDb;
using System.IO;
// ...

SnapDocumentServer server = new SnapDocumentServer();
server.SnapMailMergeRecordStarted += server_SnapMailMergeRecordStarted;
server.SnapMailMergeRecordFinished += server_SnapMailMergeRecordFinished;
server.LoadDocument(templateFileName);
object dataSource = CreateDataSource();
server.Document.DataSource = dataSource;
Console.WriteLine("Performing mail merge... ");
server.SnapMailMerge(outputFileName, DocumentFormat.Rtf);
```

See Also
[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.SnapMailMerge Overload List](#)
[Snap Report API](#)
Developer Guidelines
Snap Mail Merge

Starts rendering a mail-merge document and saving it to a stream in the specified format.

Namespace: [DevExpress.Snap](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SnapMailMerge(  
    ByVal stream As Stream,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void SnapMailMerge(  
    Stream stream,  
    DocumentFormat format  
)
```

Parameters

stream
A [System.IO.Stream](#), containing the document bytes.
format
A DevExpress.XtraRichEdit.DocumentFormat structure, specifying the document format.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E5078>.

The following code uses the Snap Report API to generate a mail-merge report in the context of the [DevExpress Office File API](#). It is a console application that creates a document and then exports it to an RTF file.

To use the [SnapDocumentServer](#), the project requires references to the following assemblies:

- DevExpress.Data.v18.1.dll
- DevExpress.Docs.v18.1.dll
- DevExpress.Office.v18.1.Core.dll

- DevExpress.RichEdit.v18.1.Core.dll
- DevExpress.Snap.v18.1.Core.dll

To perform mail merge, the application requires a template - a file in the native .SNX format which contains the document layout, specific fields and data source information. The template file must be created beforehand using the Snap Design Surface. For an example of template creation, review the Lesson 3 - Create a Mail Merge Report.

The code handles the [SnapMailMergeRecordStarted](#) and [SnapMailMergeRecordFinished](#) events to make adjustments to the merged document created for the particular data record.

The server loads a mail merge template using the DevExpress.XtraRichEdit.RichEditDocumentServer.LoadDocument method. Subsequently, the code assigns the data source by specifying the DevExpress.Snap.Core.API.IDataSourceOwner.DataSource property. The mail merge operation is performed by calling the [SnapMailMerge](#) method.

```
C#
(Program.cs)
using DevExpress.Snap;
using DevExpress.Snap.Core.API;
using DevExpress.XtraRichEdit;
using MailMergeServer.nwindDataSetTableAdapters;
using System;
using System.Data.OleDb;
using System.IO;
// ...

SnapDocumentServer server = new SnapDocumentServer();
server.SnapMailMergeRecordStarted += server_SnapMailMergeRecordStarted;
server.SnapMailMergeRecordFinished += server_SnapMailMergeRecordFinished;
server.LoadDocument(templateFileName);
object dataSource = CreateDataSource();
server.Document.DataSource = dataSource;
Console.WriteLine("Performing mail merge... ");
server.SnapMailMerge(outputFileName, DocumentFormat.Rtf);
```

See Also

[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.SnapMailMerge Overload List](#)
[Snap Report API](#)
Developer Guidelines
Snap Mail Merge

NOTE: This member is now obsolete.

This method has become obsolete. Use another SnapMailMerge method overload with appropriate parameters instead.

Obsolete. Starts rendering a mail-merge document based on the applied export options and saving it to the specified [Snap Report API](#).

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SnapMailMerge(
    ByVal options As SnapMailMergeExportOptions,
    ByVal targetDocumentServer As ISnapDocumentServer
)
```

C#

```
public void SnapMailMerge(
    SnapMailMergeExportOptions options,
    ISnapDocumentServer targetDocumentServer
)
```

Parameters

options

An object implementing the DevExpress.Snap.Core.Options.SnapMailMergeExportOptions interface.

targetDocumentServer

A [SnapDocumentServer](#) object.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.SnapMailMerge Overload List](#)

[Snap Report API](#)

Developer Guidelines

Snap Mail Merge

Starts rendering a mail-merge document based on the applied export options and saving it to the specified target document.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SnapMailMerge(  
    ByVal options As SnapMailMergeExportOptions,  
    ByVal targetDocument As SnapDocument  
)
```

C#

```
public void SnapMailMerge(  
    SnapMailMergeExportOptions options,  
    SnapDocument targetDocument  
)
```

Parameters

options

An object implementing the DevExpress.Snap.Core.Options.SnapMailMergeExportOptions interface.

targetDocument

An object implementing the DevExpress.Snap.Core.API.SnapDocument interface, storing the resulting document.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.SnapMailMerge Overload List](#)

[Snap Report API](#)

Developer Guidelines

Snap Mail Merge

Starts rendering a mail-merge document and saving it to a file in the specified format.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SnapMailMerge(  
    ByVal fileName As String,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void SnapMailMerge(  
    string fileName,  
    DocumentFormat format  
)
```

Parameters

fileName

A [System.String](#) value, specifying the file name.

format

A DevExpress.XtraRichEdit.DocumentFormat structure, specifying the document format.

Example

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E5078>.

The following code uses the Snap Report API to generate a mail-merge report in the context of the [DevExpress Office File API](#).

It is a console application that creates a document and then exports it to an RTF file.

To use the [SnapDocumentServer](#), the project requires references to the following assemblies:

- DevExpress.Data.v18.1.dll
- DevExpress.Docs.v18.1.dll
- DevExpress.Office.v18.1.Core.dll
- DevExpress.RichEdit.v18.1.Core.dll
- DevExpress.Snap.v18.1.Core.dll

To perform mail merge, the application requires a template - a file in the native .SNX format which contains the document layout, specific fields and data source information. The template file must be created beforehand using the Snap Design Surface. For an example of template creation, review the Lesson 3 - Create a Mail Merge Report.

The code handles the [SnapMailMergeRecordStarted](#) and [SnapMailMergeRecordFinished](#) events to make adjustments to the merged document created for the particular data record.

The server loads a mail merge template using the DevExpress.XtraRichEdit.RichEditDocumentServer.LoadDocument method. Subsequently, the code assigns the data source by specifying the DevExpress.Snap.Core.API.IDataSourceOwner.DataSource property. The mail merge operation is performed by calling the [SnapMailMerge](#) method.

C#

```
(Program.cs)
using DevExpress.Snap;
using DevExpress.Snap.Core.API;
using DevExpress.XtraRichEdit;
using MailMergeServer.nwindDataSetTableAdapters;
using System;
using System.Data.OleDb;
using System.IO;
// ...

SnapDocumentServer server = new SnapDocumentServer();
server.SnapMailMergeRecordStarted += server_SnapMailMergeRecordStarted;
server.SnapMailMergeRecordFinished += server_SnapMailMergeRecordFinished;
server.LoadDocument(templateFileName);
object dataSource = CreateDataSource();
server.Document.DataSource = dataSource;
Console.WriteLine("Performing mail merge... ");
server.SnapMailMerge(outputFileName, DocumentFormat.Rtf);
```

See Also

[SnapDocumentServer Class](#)
[SnapDocumentServer Members](#)
[DevExpress.Snap Namespace](#)
[SnapDocumentServer.SnapMailMerge Overload List](#)
[Snap Report API](#)

Developer Guidelines

Snap Mail Merge

Starts rendering a mail-merge document based on the applied export options and saving it to a file in the specified format.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub SnapMailMerge(  
    ByVal options As SnapMailMergeExportOptions,  
    ByVal fileName As String,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void SnapMailMerge(  
    SnapMailMergeExportOptions options,  
    string fileName,  
    DocumentFormat format  
)
```

Parameters

options

An object implementing the DevExpress.Snap.Core.Options.SnapMailMergeExportOptions interface.

fileName

A [System.String](#) value, specifying the file name.

format

A DevExpress.XtraRichEdit.DocumentFormat structure, specifying the document format.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.SnapMailMerge Overload List](#)

[Snap Report API](#)

Developer Guidelines

Snap Mail Merge

Starts rendering a mail-merge document based on the applied export options and saving it to a stream in the specified format.

Namespace: [DevExpress.Snap](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub SnapMailMerge(  
    ByVal options As SnapMailMergeExportOptions,  
    ByVal stream As Stream,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void SnapMailMerge(  
    SnapMailMergeExportOptions options,  
    Stream stream,  
    DocumentFormat format  
)
```

Parameters

options

An object implementing the DevExpress.Snap.Core.Options.SnapMailMergeExportOptions interface.

stream

A [System.IO.Stream](#), containing the document bytes.

format

A DevExpress.XtraRichEdit.DocumentFormat structure, specifying the document format.

See Also

[SnapDocumentServer Class](#)

[SnapDocumentServer Members](#)

[DevExpress.Snap Namespace](#)

[SnapDocumentServer.SnapMailMerge Overload List](#)

[Snap Report API](#)

Developer Guidelines

Snap Mail Merge

DevExpress.Spreadsheet

Contains classes and interfaces that implement basic spreadsheet functionality.

Classes

	Class	Description
	Workbook	A non-visual component providing complete spreadsheet functionality.
	WorkbookExtensions	Defines extension methods for objects exposing the IWorkbook interface.
	WorksheetExtensions	Defines extension methods for the DevExpress.Spreadsheet.Worksheet class.

Workbook Class

A non-visual component providing complete spreadsheet functionality.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class Workbook Inherits Object, IWorkbook,
    ISpreadsheetComponent,
    IBatchUpdateable,
    IServiceContainer,
    IServiceProvider,
    ISupportsContentChanged,
    IPrintable,
    IBasePrintable,
    ExternalWorkbook,
    IDisposable
```

C#

```
public class Workbook : object, IWorkbook,
    ISpreadsheetComponent,
    IBatchUpdateable,
    IServiceContainer,
    IServiceProvider,
    ISupportsContentChanged,
    IPrintable,
    IBasePrintable,
    ExternalWorkbook,
    IDisposable
```

Remarks

A **Workbook** class specifies the root object of the non-visual spreadsheet engine. It provides a comprehensive set of properties and methods required to create, load, modify, print and save the corresponding workbook without user interaction.

Use of these methods in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Refer to the [DevExpress Subscription](#) page for pricing information.

For details, see the [Workbook](#) topic and the [Examples](#) section.

Inheritance Hierarchy

[Object](#)

Workbook

See Also

[Workbook Members](#)


[DevExpress.Spreadsheet Namespace](#)

Workbook Members

A non-visual component providing complete spreadsheet functionality.

The following tables list the members exposed by the [Workbook](#) type.

Public Constructors

	Name	Description
	Workbook	Initializes a new instance of the Workbook class with the default settings.

Public Properties

	Name	Description
	ChartSheets	Provides access to a collection of chart sheets contained in the workbook . Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Clipboard	Provides access to the object used for working with the system clipboard. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CurrentAuthor	Gets the system username. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CustomFunctions	Provides access to a collection of custom functions in a workbook Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CustomXmlParts	Provides access to the collection of custom XML parts in the document. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DefinedNames	Gets the collection of defined names whose scope is the current workbook . Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

	DocumentProperties	<p>Provides access to the document properties associated with a workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	DocumentSettings	<p>Provides access to the settings that specify how the calculation is performed and what reference style is used.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ExternalWorkbooks	<p>Provides access to the collection of source workbooks used for creating external references in the current workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	FormulaEngine	<p>Provides access to a FormulaEngine object to parse or evaluate a formula.</p>
	Functions	<p>Provides access to the built-in functions in a workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	GlobalCustomFunctions	<p>Provides access to a collection of custom functions which are not limited in scope to the workbook in which the functions reside.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	HasMacros	<p>Determines whether the workbook has VBA projects (macros).</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	History	<p>Provides access to the history of operations performed in a workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	IsDisposed	<p>Gets whether a workbook has been disposed of.</p>

		Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	IsProtected	Gets whether the workbook is protected.
	IsUpdateLocked	Gets whether the Workbook object has been locked for updating. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	MailMergeDataMember	Gets or sets a specific data member in a data source that contains several tables or members. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	MailMergeDataSource	Gets or sets the data source for the mail merge. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	MailMergeOptions	Provides access to the mail merge options.
	MailMergeParameters	Provides access to a collection of parameters for queries used to obtain data in mail merge.
	Model	For internal use.
	Modified	Gets or sets whether the workbook content was modified since it was last saved. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Options	Provides access to the variety of options which can be specified for the workbook. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Path	Gets the file name into which the workbook is saved or from which it is loaded. Use of this property in production code requires a license to the DevExpress

		Office File API or the DevExpress Universal Subscription.
	PivotCaches	<p>Returns a <code>DevExpress.Spreadsheet.PivotCacheC</code> collection that represents all the PivotTable caches in the specified workbook.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Range	<p>Provides access to the cell range in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	RealTimeData	<p>Provides access to an object that is used to manually update real-time data and reconnect to data servers.</p>
	Sheets	<p>Provides access to a collection of all sheets contained in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Styles	<p>Provides access to the collection of cell styles in the current workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	TableStyles	<p>Provides access to the collection of styles to format tables in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Tag	<p>Gets or sets the data associated with a Workbook object.</p>
	Unit	<p>Gets or sets a unit of measure used in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Worksheets	<p>Gets a collection of worksheets contained in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress</p>

Office File API or the DevExpress Universal Subscription.

Public Methods

	Name	Description
	AddService	Overloaded. Adds the specified service to the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeginUpdate	Locks the Workbook object until the EndUpdate or CancelUpdate method is called. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Calculate	Overloaded. Forces recalculation of the workbook. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CalculateFull	Forces a full calculation of the data in a workbook.
	CalculateFullRebuild	Forces a full calculation of the data and rebuilds the dependencies.
	CancelUpdate	Unlocks the Workbook object after it has been locked by the BeginUpdate method, without causing an immediate update. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CreateNewDocument	Creates and loads a new empty workbook. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Dispose	Releases resources associated with a Workbook instance. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	EndUpdate	Unlocks the Workbook object.

		Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Evaluate	Overloaded. Evaluates the specified formula. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ExportToHtml	Overloaded. Exports the specified range to the specified stream in HTML format. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ExportToPdf	Overloaded. Exports the workbook to the specified stream in PDF format using the specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GenerateMailMergeDocuments	Performs a mail merge and returns the collection of resulting workbooks. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetService	Gets the service object of the specified type. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetService<T>	Gets the specified service.

		Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	LoadDocument	Overloaded. Loads the document from a System.Byte[] array.
	Print	Overloaded. Prints the document using the specified printer settings. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Protect	Protects the structure and windows of a workbook.
 S	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveService	Overloaded. Removes the service of the specified type from the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ReplaceService<T>	Performs a service substitution. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	SaveDocument	Overloaded. Saves a document to an array of bytes in the specified format.
	Search	Overloaded. Performs a search in the current document using specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
	Unprotect	Removes protection from a workbook.

Public Events

	Name	Description
--	------	-------------

	ActiveSheetChanged	Occurs after an active worksheet in a workbook has been changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ActiveSheetChanging	Occurs when an active worksheet in a workbook is about to be changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforeExport	Occurs before the document is saved (exported to a certain format). Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforeImport	Occurs before a document is loaded (imported from an external source). Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforePrintSheet	Occurs before printing a workbook.
	ClipboardDataObtained	Occurs after data on the clipboard is obtained and recognized, but before data is actually pasted. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ClipboardDataPasted	Occurs after data has been pasted from the clipboard onto a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ClipboardDataPasting	Occurs before data is pasted into destination cells. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ColumnsInserted	Occurs after new columns have been added to a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

	ColumnsInserting	Occurs before inserting new columns into a worksheet.
	ColumnsRemoved	Occurs after columns have been deleted from a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ColumnsRemoving	Occurs before deleting columns from a worksheet.
	CommentInserted	Occurs when a comment is inserted.
	CommentInserting	Occurs before inserting a comment.
	CommentRemoved	Occurs when a comment is deleted.
	CommentRemoving	Occurs before deleting a comment.
	ContentChanged	Occurs when the workbook's undo history changes. By default, undo history is not logged for the Workbook instance, so the event is not raised. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CopiedRangePasted	Occurs after the range content has been pasted into target cells. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CopiedRangePasting	Occurs before the range content is pasted into target cells. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CustomAssemblyLoading	Occurs before a custom assembly is loaded for use as the Entity Framework data source during mail merge and allows cancelling loading. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DefinedNameConflictResolving	Occurs when the formula or sheet being moved or copied contains a defined name which already exists on the destination worksheet or workbook. Use of this event in production code requires a license to the DevExpress

		Office File API or the DevExpress Universal Subscription.
	DocumentClosing	Occurs when a document that has not yet been saved is about to be closed. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DocumentLoaded	Occurs after a document is loaded. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DocumentSaved	Occurs after a document has been saved. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	EmptyDocumentCreated	Occurs when a new document is created. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	EncryptedFileIntegrityCheckFailed	Raises when the encrypted file did not pass the data integrity verification.
	EncryptedFilePasswordRequest	Raises when the DevExpress.XtraSpreadsheet.Import.WorkbookImportOptions.Password property is not set or contains a wrong password.
	InitializeDocument	Occurs before a document is loaded. Handle this event to set initial document settings. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	InvalidFormatException	Fires when the supplied data could not be recognized as data in the assumed format for import. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ModifiedChanged	Occurs when the value of the Modified property is changed. Use of this event in production code requires a license to the DevExpress

		Office File API or the DevExpress Universal Subscription.
	PanesFrozen	<p>Occurs after a worksheet area has been frozen.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	PanesUnfrozen	<p>Occurs after a frozen worksheet area has been unlocked.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	RangeCopied	<p>Occurs after the range content has been copied.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	RangeCopying	<p>Occurs before a cell range is copied in a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	RowsInserted	<p>Occurs after new rows have been added to a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	RowsInserting	Occurs before inserting new rows into a worksheet.
	RowsRemoved	<p>Occurs after rows have been deleted from a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	RowsRemoving	Occurs before deleting rows from a worksheet.
	SchemaChanged	<p>Occurs when a worksheet or defined name is renamed, inserted or deleted.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>

	ScrollPositionChanged	<p>Occurs when the scroll position changes in a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SelectionChanged	<p>Fires in response to changing cell selection in a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ShapeInserted	<p>Occurs after a drawing object has been inserted into a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ShapeRemoved	<p>Occurs after a drawing object has been removed from a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ShapeRemoving	<p>Occurs before a drawing object is removed from a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ShapesCopying	<p>Occurs before a drawing object is copied in a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SheetInserted	<p>Occurs after a new worksheet has been added to a workbook.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SheetRemoved	<p>Occurs after a worksheet has been removed from a workbook.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SheetRemoving	<p>Occurs before a worksheet is removed from a workbook.</p>

	SheetRenamed	<p>Occurs after a worksheet has been renamed.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SheetRenaming	<p>Occurs when a worksheet is about to be renamed.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	UnitChanged	<p>Fires after a unit of measurement used in the workbook is changed.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	UnitChanging	<p>Fires before a unit of measurement used within the workbook is changed.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ValidateCustomSqlQuery	<p>Allows validation of the custom SQL query created using the Data Source Wizard.</p>

See Also
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)

Workbook Constructor

Initializes a new instance of the [Workbook](#) class with the default settings.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public Workbook()
```

See Also

[Workbook Class](#)

[Workbook Members](#)


[DevExpress.Spreadsheet Namespace](#)

Workbook Properties

A non-visual component providing complete spreadsheet functionality.

The following tables list the members exposed by the [Workbook](#) type.

Public Constructors

	Name	Description
	Workbook	Initializes a new instance of the Workbook class with the default settings.

Public Properties

	Name	Description
	ChartSheets	Provides access to a collection of chart sheets contained in the workbook . Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Clipboard	Provides access to the object used for working with the system clipboard. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CurrentAuthor	Gets the system username. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CustomFunctions	Provides access to a collection of custom functions in a workbook Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CustomXmlParts	Provides access to the collection of custom XML parts in the document. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DefinedNames	Gets the collection of defined names whose scope is the current workbook . Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

	DocumentProperties	<p>Provides access to the document properties associated with a workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	DocumentSettings	<p>Provides access to the settings that specify how the calculation is performed and what reference style is used.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ExternalWorkbooks	<p>Provides access to the collection of source workbooks used for creating external references in the current workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	FormulaEngine	<p>Provides access to a FormulaEngine object to parse or evaluate a formula.</p>
	Functions	<p>Provides access to the built-in functions in a workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	GlobalCustomFunctions	<p>Provides access to a collection of custom functions which are not limited in scope to the workbook in which the functions reside.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	HasMacros	<p>Determines whether the workbook has VBA projects (macros).</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	History	<p>Provides access to the history of operations performed in a workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	IsDisposed	<p>Gets whether a workbook has been disposed of.</p>

		Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	IsProtected	Gets whether the workbook is protected.
	IsUpdateLocked	Gets whether the Workbook object has been locked for updating. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	MailMergeDataMember	Gets or sets a specific data member in a data source that contains several tables or members. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	MailMergeDataSource	Gets or sets the data source for the mail merge. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	MailMergeOptions	Provides access to the mail merge options.
	MailMergeParameters	Provides access to a collection of parameters for queries used to obtain data in mail merge.
	Model	For internal use.
	Modified	Gets or sets whether the workbook content was modified since it was last saved. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Options	Provides access to the variety of options which can be specified for the workbook. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Path	Gets the file name into which the workbook is saved or from which it is loaded. Use of this property in production code requires a license to the DevExpress

		Office File API or the DevExpress Universal Subscription.
	PivotCaches	<p>Returns a <code>DevExpress.Spreadsheet.PivotCacheC</code> collection that represents all the PivotTable caches in the specified workbook.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Range	<p>Provides access to the cell range in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	RealTimeData	<p>Provides access to an object that is used to manually update real-time data and reconnect to data servers.</p>
	Sheets	<p>Provides access to a collection of all sheets contained in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Styles	<p>Provides access to the collection of cell styles in the current workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	TableStyles	<p>Provides access to the collection of styles to format tables in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Tag	<p>Gets or sets the data associated with a Workbook object.</p>
	Unit	<p>Gets or sets a unit of measure used in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Worksheets	<p>Gets a collection of worksheets contained in the workbook.</p> <p>Use of this property in production code requires a license to the DevExpress</p>

Office File API or the DevExpress Universal Subscription.

Public Methods

	Name	Description
	AddService	Overloaded. Adds the specified service to the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeginUpdate	Locks the Workbook object until the EndUpdate or CancelUpdate method is called. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Calculate	Overloaded. Forces recalculation of the workbook. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CalculateFull	Forces a full calculation of the data in a workbook.
	CalculateFullRebuild	Forces a full calculation of the data and rebuilds the dependencies.
	CancelUpdate	Unlocks the Workbook object after it has been locked by the BeginUpdate method, without causing an immediate update. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CreateNewDocument	Creates and loads a new empty workbook. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Dispose	Releases resources associated with a Workbook instance. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	EndUpdate	Unlocks the Workbook object.

		Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Evaluate	Overloaded. Evaluates the specified formula. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ExportToHtml	Overloaded. Exports the specified range to the specified stream in HTML format. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ExportToPdf	Overloaded. Exports the workbook to the specified stream in PDF format using the specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GenerateMailMergeDocuments	Performs a mail merge and returns the collection of resulting workbooks. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetService	Gets the service object of the specified type. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetService<T>	Gets the specified service.

		Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	LoadDocument	Overloaded. Loads the document from a System.Byte[] array.
	Print	Overloaded. Prints the document using the specified printer settings. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Protect	Protects the structure and windows of a workbook.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveService	Overloaded. Removes the service of the specified type from the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ReplaceService<T>	Performs a service substitution. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	SaveDocument	Overloaded. Saves a document to an array of bytes in the specified format.
	Search	Overloaded. Performs a search in the current document using specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
	Unprotect	Removes protection from a workbook.

Public Events

	Name	Description
--	------	-------------

	ActiveSheetChanged	Occurs after an active worksheet in a workbook has been changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ActiveSheetChanging	Occurs when an active worksheet in a workbook is about to be changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforeExport	Occurs before the document is saved (exported to a certain format). Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforeImport	Occurs before a document is loaded (imported from an external source). Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforePrintSheet	Occurs before printing a workbook.
	ClipboardDataObtained	Occurs after data on the clipboard is obtained and recognized, but before data is actually pasted. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ClipboardDataPasted	Occurs after data has been pasted from the clipboard onto a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ClipboardDataPasting	Occurs before data is pasted into destination cells. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ColumnsInserted	Occurs after new columns have been added to a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

	ColumnsInserting	Occurs before inserting new columns into a worksheet.
	ColumnsRemoved	Occurs after columns have been deleted from a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ColumnsRemoving	Occurs before deleting columns from a worksheet.
	CommentInserted	Occurs when a comment is inserted.
	CommentInserting	Occurs before inserting a comment.
	CommentRemoved	Occurs when a comment is deleted.
	CommentRemoving	Occurs before deleting a comment.
	ContentChanged	Occurs when the workbook's undo history changes. By default, undo history is not logged for the Workbook instance, so the event is not raised. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CopiedRangePasted	Occurs after the range content has been pasted into target cells. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CopiedRangePasting	Occurs before the range content is pasted into target cells. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CustomAssemblyLoading	Occurs before a custom assembly is loaded for use as the Entity Framework data source during mail merge and allows cancelling loading. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DefinedNameConflictResolving	Occurs when the formula or sheet being moved or copied contains a defined name which already exists on the destination worksheet or workbook. Use of this event in production code requires a license to the DevExpress

		Office File API or the DevExpress Universal Subscription.
	DocumentClosing	Occurs when a document that has not yet been saved is about to be closed. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DocumentLoaded	Occurs after a document is loaded. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DocumentSaved	Occurs after a document has been saved. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	EmptyDocumentCreated	Occurs when a new document is created. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	EncryptedFileIntegrityCheckFailed	Raises when the encrypted file did not pass the data integrity verification.
	EncryptedFilePasswordRequest	Raises when the DevExpress.XtraSpreadsheet.Import.WorkbookImportOptions.Password property is not set or contains a wrong password.
	InitializeDocument	Occurs before a document is loaded. Handle this event to set initial document settings. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	InvalidFormatException	Fires when the supplied data could not be recognized as data in the assumed format for import. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ModifiedChanged	Occurs when the value of the Modified property is changed. Use of this event in production code requires a license to the DevExpress

		Office File API or the DevExpress Universal Subscription.
	PanesFrozen	Occurs after a worksheet area has been frozen. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	PanesUnfrozen	Occurs after a frozen worksheet area has been unlocked. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RangeCopied	Occurs after the range content has been copied. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RangeCopying	Occurs before a cell range is copied in a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RowsInserted	Occurs after new rows have been added to a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RowsInserting	Occurs before inserting new rows into a worksheet.
	RowsRemoved	Occurs after rows have been deleted from a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RowsRemoving	Occurs before deleting rows from a worksheet.
	SchemaChanged	Occurs when a worksheet or defined name is renamed, inserted or deleted. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

	ScrollPositionChanged	<p>Occurs when the scroll position changes in a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SelectionChanged	<p>Fires in response to changing cell selection in a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ShapeInserted	<p>Occurs after a drawing object has been inserted into a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ShapeRemoved	<p>Occurs after a drawing object has been removed from a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ShapeRemoving	<p>Occurs before a drawing object is removed from a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ShapesCopying	<p>Occurs before a drawing object is copied in a worksheet.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SheetInserted	<p>Occurs after a new worksheet has been added to a workbook.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SheetRemoved	<p>Occurs after a worksheet has been removed from a workbook.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SheetRemoving	<p>Occurs before a worksheet is removed from a workbook.</p>

	SheetRenamed	<p>Occurs after a worksheet has been renamed.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	SheetRenaming	<p>Occurs when a worksheet is about to be renamed.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	UnitChanged	<p>Fires after a unit of measurement used in the workbook is changed.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	UnitChanging	<p>Fires before a unit of measurement used within the workbook is changed.</p> <p>Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ValidateCustomSqlQuery	<p>Allows validation of the custom SQL query created using the Data Source Wizard.</p>

See Also[Workbook Members](#)[DevExpress.Spreadsheet Namespace](#)

ChartSheets Property

Provides access to a collection of chart sheets contained in the [workbook](#).

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property ChartSheets As ChartSheetCollection
```

C#

```
public ChartSheetCollection ChartSheets { get; }
```

Property Value

A DevExpress.Spreadsheet.ChartSheetCollection object that is a collection of chart sheets.

Remarks

Use the **ChartSheets** property to get access to the DevExpress.Spreadsheet.ChartSheetCollection collection, which stores all chart sheets specified in a workbook. An individual DevExpress.Spreadsheet.ChartSheet can be accessed by its name or index in the collection.

To create a new chart sheet, use the DevExpress.Spreadsheet.ChartSheetCollection.Add method. To move an existing chart to a chart sheet, use the chart's DevExpress.Spreadsheet.Charts.ChartObject.MoveToNewChartSheet method.

Use the DevExpress.Spreadsheet.ChartSheetCollection.Remove or DevExpress.Spreadsheet.ChartSheetCollection.RemoveAt method to remove a chart sheet from the collection.

The DevExpress.Spreadsheet.SheetCollection collection contains all sheets in a workbook (both chart sheets and worksheets). Use this collection when you need to access a sheet of any type by its name or index in the document.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Clipboard Property

Provides access to the object used for working with the system clipboard.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Clipboard As IClipboardManager
```

C#

```
public IClipboardManager Clipboard { get; }
```

Property Value

An object implementing the DevExpress.Spreadsheet.IClipboardManager interface.

Remarks

Use the **Clipboard** property to access the DevExpress.Spreadsheet.IClipboardManager object that allows you to transfer data from the workbook to another document or application by copying values of the selected cells to the clipboard.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CurrentAuthor Property

Gets the system username.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property CurrentAuthor As String
```

C#

```
public string CurrentAuthor { get; }
```

Property Value

A [System.String](#) value that specifies the username of the person currently logged on the operating system.

Remarks

You can use the **CurrentAuthor** property to specify an author of cell comments (see the [How To: Add a Comment To a Cell](#) example).

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CustomFunctions Property

Provides access to a collection of custom functions in a workbook

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property CustomFunctions As CustomFunctionCollection
```

C#

```
public CustomFunctionCollection CustomFunctions { get; }
```

Property Value

A DevExpress.Spreadsheet.Functions.CustomFunctionCollection collection of custom functions.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

DevExpress.Spreadsheet.Functions.ICustomFunction

CustomXmlParts Property

Provides access to the collection of custom XML parts in the document.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property CustomXmlParts As CustomXmlPartCollection
```

C#

```
public CustomXmlPartCollection CustomXmlParts { get; }
```

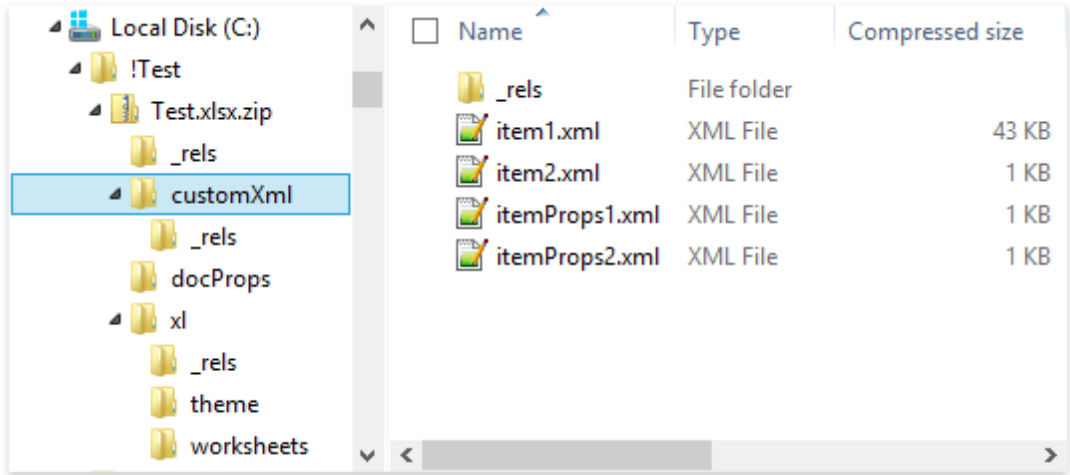
Property Value

A CustomXmlPartCollection object that is the collection of embedded arbitrary XML data (custom XML parts).

Remarks

A document in OpenXml (.xlsx) format consists of XML files within folders packed in a ZIP archive. Most of the XML files are built-in parts that define the document structure and hold its content. However, documents can also contain custom XML parts, which you can use to store arbitrary XML data.

The folder that contains custom XML parts is located in the document structure as shown in the following picture. There are two custom XML parts clearly identified as item1 and item2.



You can use the DevExpress.Spreadsheet.CustomXmlPartCollection.Add method to store an XML string or a [System.Xml.XmlDocument](#) document as the document's custom XML part.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

DefinedNames Property

Gets the collection of [defined names](#) whose scope is the current [workbook](#).

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property DefinedNames As DefinedNameCollection
```

C#

```
public DefinedNameCollection DefinedNames { get; }
```

Property Value

A DevExpress.Spreadsheet.DefinedNameCollection collection of a worksheet's defined names.

Remarks

Each [defined name](#) has a scope - an area (an individual worksheet or entire workbook) within which a name is recognized and can be used without qualification. So, each worksheet contained in the workbook and the entire workbook has it's own collection of defined names that can be accessed by the DevExpress.Spreadsheet.Worksheet.DefinedNames or **DefinedNames** property, respectively.

Use members of the DevExpress.Spreadsheet.DefinedNameCollection object to create new defined names for [cells](#) and [formulas](#), access an individual defined name by its index in the collection or name, remove names from the collection, etc.

For more details on defined names, refer to the [Defined Names](#) document.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to define names for [formulas](#). To do this, call the DevExpress.Spreadsheet.DefinedNameCollection.Add method with a name to be associated with a formula and the formula string passed as parameters. Use the DevExpress.Spreadsheet.Worksheet.DefinedNames or [DefinedNames](#) property to access and modify the collection of defined names of a particular worksheet or entire workbook, depending on which scope you want to specify for a name.

C#

```
(FormulaActions.cs)
Worksheet worksheet1 = workbook.Worksheets["Sheet1"];
Worksheet worksheet2 = workbook.Worksheets["Sheet2"];
// Create a name for a formula that sums up the values of all cells included in the "A1:C3" range of the
// The scope of this name will be limited by the "Sheet1" worksheet.
worksheet1.DefinedNames.Add("Range_Sum", "=SUM(Sheet1!$A$1:$C$3)");
// Create a name for a formula that doubles the value resulting from the "Range_Sum" named formula and
// make this name available within the entire workbook.
workbook.DefinedNames.Add("Range_DoubleSum", "=2*Sheet1!Range_Sum");
// Create formulas that use other formulas with the specified names.
worksheet2.Cells["C2"].Formula = "=Sheet1!Range_Sum";
worksheet2.Cells["C3"].Formula = "=Range_DoubleSum";
worksheet2.Cells["C4"].Formula = "=Range_DoubleSum + 100";
```

Visual Basic

```
(FormulaActions.vb)
Dim worksheet1 As Worksheet = workbook.Worksheets("Sheet1")
Dim worksheet2 As Worksheet = workbook.Worksheets("Sheet2")
' Create a name for a formula that sums up the values of all cells included in the "A1:C3" range of the "Sheet1" worksheet.
' The scope of this name will be limited by the "Sheet1" worksheet.
worksheet1.NamedRanges.Add("Range_Sum", "=SUM(Sheet1!$A$1:$C$3)")
' Create a name for a formula that doubles the value resulting from the "Range_Sum" named formula and
' make this name available within the entire workbook.
workbook.NamedRanges.Add("Range_DoubleSum", "=2*Sheet1!Range_Sum")
' Create formulas that use other formulas with the specified names.
worksheet2.Cells("C2").Formula = "=Sheet1!Range_Sum"
worksheet2.Cells("C3").Formula = "=Range_DoubleSum"
worksheet2.Cells("C4").Formula = "=Range_DoubleSum + 100"
```

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[DevExpress.Spreadsheet.Worksheet.NamedRanges](#)

[Named Ranges](#)

[How to: Create a Named Range of Cells](#)

[How to: Create Named Formulas](#)

DocumentProperties Property

Provides access to the document properties associated with a workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property DocumentProperties As DocumentProperties
```

C#

```
public DocumentProperties DocumentProperties { get; }
```

Property Value

A DevExpress.Spreadsheet.DocumentProperties object containing information about a workbook.

Remarks

Use the **DocumentProperties** property to specify the built-in document properties that contain basic information about the spreadsheet document (such as DevExpress.Spreadsheet.DocumentProperties.Title, DevExpress.Spreadsheet.DocumentProperties.Author, DevExpress.Spreadsheet.DocumentProperties.Subject, DevExpress.Spreadsheet.DocumentProperties.Description etc.). You can also create your own custom document properties using the DevExpress.Spreadsheet.DocumentProperties.Custom property.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

DevExpress.Spreadsheet.DocumentProperties.Custom

DocumentSettings Property

Provides access to the settings that specify how the calculation is performed and what reference style is used.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property DocumentSettings As DocumentSettings
```

C#

```
public DocumentSettings DocumentSettings { get; }
```

Property Value

A DevExpress.Spreadsheet.DocumentSettings object containing settings for the reference style and calculation options.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

ExternalWorkbooks Property

Provides access to the collection of source workbooks used for creating external references in the current workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property ExternalWorkbooks As ExternalWorkbookCollection
```

C#

```
public ExternalWorkbookCollection ExternalWorkbooks { get; }
```

Property Value

An ExternalWorkbookCollection object.

Remarks

To use cell references and defined names contained in another workbook, you should add a workbook to the **ExternalWorkbooks** collection.

The following code snippet creates a workbook, populates it with random data by importing a data table and adds to the collection of external workbooks.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T220356>.

C#

```
(Form1.cs)
Workbook externalWorkbook = new Workbook();
externalWorkbook.Options.Save.CurrentFileName = "ExternalDocument.xlsx";
// Check whether the external workbook is already referenced.
foreach (IWorkbook item in myWorkbook.ExternalWorkbooks)
{
    if (item.Options.Save.CurrentFileName == externalWorkbook.Options.Save.CurrentFileName)
        return;
}
externalWorkbook.Worksheets[0].Import(CreateDataTable(10), false, 0, 0);
externalWorkbook.SaveDocument("ExternalDocument.xlsx");
myWorkbook.ExternalWorkbooks.Add(externalWorkbook);
```

Visual Basic

```
(Form1.vb)
Dim externalWorkbook As New Workbook()
externalWorkbook.Options.Save.CurrentFileName = "ExternalDocument.xlsx"
' Check whether the external workbook is already referenced.
For Each item As IWorkbook In myWorkbook.ExternalWorkbooks
    If item.Options.Save.CurrentFileName = externalWorkbook.Options.Save.CurrentFileName Then
        Return
    End If
Next item
externalWorkbook.Worksheets(0).Import(CreateDataTable(10), False, 0, 0)
externalWorkbook.SaveDocument("ExternalDocument.xlsx")
myWorkbook.ExternalWorkbooks.Add(externalWorkbook)
```

Subsequently you can reference this workbook from the current workbook, as illustrated in the code below.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=T220356>.

C#

```
(Form1.cs)
if (myWorkbook.ExternalWorkbooks.Count == 0)
{
    return;
}
IWorkbook extWorkbook = (IWorkbook)myWorkbook.ExternalWorkbooks[0];
string extWorkbookName = extWorkbook.Options.Save.CurrentFileName;
string sFormula = String.Format("={0}Sheet1!A1", extWorkbookName);
myWorkbook.Worksheets[0].Cells["A1"].Formula = sFormula;
myWorkbook.SaveDocument("Test.xlsx");
System.Diagnostics.Process.Start("Test.xlsx");
```

Visual Basic

```
(Form1.vb)
If myWorkbook.ExternalWorkbooks.Count = 0 Then
    Return
End If
Dim extWorkbook As IWorkbook = DirectCast(myWorkbook.ExternalWorkbooks(0),
Dim extWorkbookName As String = extWorkbook.Options.Save.CurrentFileName
Dim sFormula As String = String.Format("={0}Sheet1!A1", extWorkbookName)
myWorkbook.Worksheets(0).Cells("A1").Formula = sFormula
myWorkbook.SaveDocument("Test.xlsx")
System.Diagnostics.Process.Start("Test.xlsx")
```

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

FormulaEngine Property

Provides access to a FormulaEngine object to parse or evaluate a formula.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property FormulaEngine As FormulaEngine
```

C#

```
public FormulaEngine FormulaEngine { get; }
```

Property Value

A DevExpress.Spreadsheet.Formulas.FormulaEngine object that serves as a formula calculator and parser.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Functions Property

Provides access to the built-in functions in a workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Functions As WorkbookFunctions
```

C#

```
public WorkbookFunctions Functions { get; }
```

Property Value

An object implementing the `DevExpress.Spreadsheet.Functions.WorkbookFunctions` interface.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Functions](#)

GlobalCustomFunctions Property

Provides access to a collection of custom functions which are not limited in scope to the workbook in which the functions reside.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property GlobalCustomFunctions As CustomFunctionCollection
```

C#

```
public CustomFunctionCollection GlobalCustomFunctions { get; }
```

Property Value

A DevExpress.Spreadsheet.Functions.CustomFunctionCollection collection containing custom functions.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

HasMacros Property

Determines whether the workbook has VBA projects (macros).

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property HasMacros As Boolean
```

C#

```
public bool HasMacros { get; }
```

Property Value

true, if a workbook has macros; otherwise, **false**.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

History Property

Provides access to the history of operations performed in a workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property History As SpreadsheetHistory
```

C#

```
public SpreadsheetHistory History { get; }
```

Property Value

A DevExpress.Spreadsheet.SpreadsheetHistory object.

Remarks

The **History** property provides access to the DevExpress.Spreadsheet.SpreadsheetHistory object that allows you to undo (the DevExpress.Spreadsheet.SpreadsheetHistory.Undo method) and redo (the DevExpress.Spreadsheet.SpreadsheetHistory.Redo method) the last actions performed, or to clear the history of operations (the DevExpress.Spreadsheet.SpreadsheetHistory.Clear method). To trace the history of changes made in a workbook, set the DevExpress.Spreadsheet.SpreadsheetHistory.IsEnabled property to **true**.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

IsDisposed Property

Gets whether a workbook has been disposed of.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property IsDisposed As Boolean
```

C#

```
public bool IsDisposed { get; }
```

Property Value

true, if the workbook is disposed of; otherwise, **false**.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

IsProtected Property

Gets whether the workbook is protected.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property IsProtected As Boolean
```

C#

```
public bool IsProtected { get; }
```

Property Value

true, if the workbook is protected; otherwise, **false**.

Remarks

Use the **IsProtected** property to check whether the workbook is already protected before applying protection with the [Protect](#) method.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Protect](#)

IsUpdateLocked Property

Gets whether the [Workbook](#) object has been locked for updating.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property IsUpdateLocked As Boolean
```

C#

```
public bool IsUpdateLocked { get; }
```

Property Value

true, if the object is locked; otherwise, **false**.

Remarks

The [BeginUpdate](#), [EndUpdate](#) and [CancelUpdate](#) methods use an internal counter to implement the update functionality. The counter's initial value is 0. The [BeginUpdate](#) method increments the counter. [EndUpdate](#) and [CancelUpdate](#) decrement the counter. The updates are enabled if the counter's value is 0. Each call to [BeginUpdate](#) must be paired with a call to [EndUpdate](#) or [CancelUpdate](#). If a call to [BeginUpdate](#) is made without a corresponding call to [EndUpdate](#)/[CancelUpdate](#) afterwards, or it's not called because an exception occurred, the object will no longer be updated. To ensure that [EndUpdate](#)/[CancelUpdate](#) is always called even if an exception occurs, use the **try...finally** statement.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[BeginUpdate](#)

[EndUpdate](#)

[CancelUpdate](#)

MailMergeDataMember Property

Gets or sets a specific data member in a data source that contains several tables or members.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property MailMergeDataMember As String
```

C#

```
public string MailMergeDataMember { get; set; }
```

Property Value

A string value specifying the data source member.

Remarks

Use the **MailMergeDataMember** property if the [MailMergeDataSource](#) specifies a dataset which contains several data tables. If the data source is represented by a data table, data view or any custom created data source object, the **MailMergeDataMember** is irrelevant.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

MailMergeDataSource Property

Gets or sets the data source for the mail merge.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property MailMergeDataSource As Object
```

C#

```
public object MailMergeDataSource { get; set; }
```

Property Value

An object that specifies the data source from which the merged data is retrieved.

Remarks

Any object that provides the IList interface can serve as the data source for the mail merge using the spreadsheet component.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

MailMergeOptions Property

Provides access to the [mail merge](#) options.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property MailMergeOptions As SpreadsheetMailMergeOptions
```

C#

```
public SpreadsheetMailMergeOptions MailMergeOptions { get; }
```

Property Value

A DevExpress.Spreadsheet.SpreadsheetMailMergeOptions object containing the mail merge options.

Remarks

Use the **MailMergeOptions** property to get access to the DevExpress.Spreadsheet.SpreadsheetMailMergeOptions object. Use the object's DevExpress.Spreadsheet.SpreadsheetMailMergeOptions.UseTemplateSheetNames property to generate worksheet names for the resulting document(s) based on the template worksheet's name.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

MailMergeParameters Property

Provides access to a collection of parameters for queries used to obtain data in mail merge.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property MailMergeParameters As ParametersCollection
```

C#

```
public ParametersCollection MailMergeParameters { get; }
```

Property Value

A DevExpress.Spreadsheet.ParametersCollection collection of the DevExpress.Spreadsheet.Parameter types.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Model Property

For internal use.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Model As DocumentModelAccessor
```

C#

```
public DocumentModelAccessor Model { get; }
```

Property Value

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Modified Property

Gets or sets whether the workbook content was modified since it was last saved.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Property Modified As [Boolean](#)

C#

```
public bool Modified { get; set; }
```

Property Value

true, if the workbook content was modified; otherwise, **false**.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Options Property

Provides access to the variety of options which can be specified for the workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Options As DocumentOptions
```

C#

```
public DocumentOptions Options { get; }
```

Property Value

A DevExpress.Spreadsheet.DocumentOptions object containing various workbook options.

Remarks

The **Options** property provides access to the object which exposes all options implemented for the workbook.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Path Property

Gets the file name into which the workbook is saved or from which it is loaded.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Path As String
```

C#

```
public string Path { get; }
```

Property Value

A [System.String](#) which specifies the current file name (including the path and the file extension).

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

PivotCaches Property

Returns a `DevExpress.Spreadsheet.PivotCacheCollection` collection that represents all the PivotTable caches in the specified workbook.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property PivotCaches As PivotCacheCollection
```

C#

```
public PivotCacheCollection PivotCaches { get; }
```

Property Value

A `DevExpress.Spreadsheet.PivotCacheCollection` that is the collection of data caches from the PivotTable reports in a workbook.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Range Property

Provides access to the [cell range](#) in the workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Range As IRangeProvider
```

C#

```
public IRangeProvider Range { get; }
```

Property Value

An object implementing the DevExpress.Spreadsheet.IRangeProvider interface.

Remarks

A cell range is a rectangular block of cells that is specified by the DevExpress.Spreadsheet.Range object. The **Range** property returns the DevExpress.Spreadsheet.IRangeProvider object whose members can be used to obtain a cell range.

- DevExpress.Spreadsheet.IRangeProvider.Item, DevExpress.Spreadsheet.IRangeProvider.Parse - obtain a cell range by its [cell reference](#) or [name](#).
- To access a cell range located in a specific worksheet of the workbook, specify this worksheet name before the cell reference, and separate them with an exclamation point (for example, *workbook.Range.Parse("Sheet2!C3:D9")*). If you do not specify the worksheet name explicitly, the cell range located on the currently active worksheet is returned.
- DevExpress.Spreadsheet.IRangeProvider.FromLTRB - obtains a cell range by the indexes of its top left and bottom right cells.

This method returns a cell range located in the worksheet that is currently active.

In addition, to get a cell range located in a specific worksheet, you can use the DevExpress.Spreadsheet.Worksheet.Range property of the corresponding worksheet object.

Example

This example demonstrates how to access ranges of cells in a worksheet. There are several ways to accomplish this.

- Use the DevExpress.Spreadsheet.Worksheet.Item property.
- Use the DevExpress.Spreadsheet.IRangeProvider.Item, DevExpress.Spreadsheet.IRangeProvider.Parse and DevExpress.Spreadsheet.IRangeProvider.FromLTRB members of the DevExpress.Spreadsheet.IRangeProvider object, accessed via the DevExpress.Spreadsheet.Worksheet.Range or [Range](#) property.

C#	
----	--

```

using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
Worksheet worksheet = workbook.Worksheets[0];
// A range that includes cells from the top left cell (A1) to the bottom right cell (B5).
Range rangeA1B5 = worksheet["A1:B5"];
// A rectangular range that includes cells from the top left cell (C4) to the bottom right cell (E7).
Range rangeC4E7 = worksheet.Range["C4:E7"];
// The C4:E7 cell range located in the "Sheet3" worksheet.
Range rangeSheet3C4E7 = workbook.Range["Sheet3!C4:E7"];
// A range that contains a single cell (E7).
Range rangeE7 = worksheet.Range["E7"];
// A range that includes the entire column A.
Range rangeColumnA = worksheet.Range["A:A"];
// A range that includes the entire row 5.
Range rangeRow5 = worksheet.Range["5:5"];
// A minimal rectangular range that includes all listed cells: C6, D9 and E7.
Range rangeC6D9E7 = worksheet.Range.Parse("C6:D9:E7");
// A rectangular range whose left column index is 0, top row index is 0,
// right column index is 3 and bottom row index is 2. This is the A1:D3 cell range.
Range rangeA1D3 = worksheet.Range.FromLTRB(0, 0, 3, 2);
// A range that includes the intersection of two ranges: C5:E10 and E9:G13.
// This is the E9:E10 cell range.
Range rangeE9E10 = worksheet.Range["C5:E10 E9:G13"];
// Create a defined name for the D20:G23 cell range.
worksheet.NamedRanges.Add("Range_Name", "Sheet1!$D$20:$G$23");
// Access a range by its defined name.
Range rangeD20G23 = worksheet.Range["Range_Name"];

```

Visual Basic

```

Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
Dim worksheet As Worksheet = workbook.Worksheets(0)
' A range that includes cells from the top left cell (A1) to the bottom right cell (B5).
Dim rangeA1B5 As Range = worksheet("A1:B5")
' A rectangular range that includes cells from the top left cell (C4) to the bottom right cell (E7).
Dim rangeC4E7 As Range = worksheet.Range("C4:E7")
' The C4:E7 cell range located in the "Sheet3" worksheet.
Dim rangeSheet3C4E7 As Range = workbook.Range("Sheet3!C4:E7")
' A range that contains a single cell (E7).
Dim rangeE7 As Range = worksheet.Range("E7")
' A range that includes the entire column A.
Dim rangeColumnA As Range = worksheet.Range("A:A")
' A range that includes the entire row 5.
Dim rangeRow5 As Range = worksheet.Range("5:5")
' A minimal rectangular range that includes all listed cells: C6, D9 and E7.
Dim rangeC6D9E7 As Range = worksheet.Range.Parse("C6:D9:E7")
' A rectangular range whose left column index is 0, top row index is 0,
' right column index is 3 and bottom row index is 2. This is the A1:D3 cell range.
Dim rangeA1D3 As Range = worksheet.Range.FromLTRB(0, 0, 3, 2)
' A range that includes the intersection of two ranges: C5:E10 and E9:G13.
' This is the E9:E10 cell range.
Dim rangeE9E10 As Range = worksheet.Range("C5:E10 E9:G13")
' Create a defined name for the D20:G23 cell range.
worksheet.NamedRanges.Add("Range_Name", "Sheet1!$D$20:$G$23")
' Access a range by its defined name.
Dim rangeD20G23 As Range = worksheet.Range("Range_Name")

```

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

DevExpress.Spreadsheet.Worksheet.Range

RealTimeData Property

Provides access to an object that is used to manually update real-time data and reconnect to data servers.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property RealTimeData As RealTimeData
```


C#

```
public RealTimeData RealTimeData { get; }
```

Property Value

An object with the DevExpress.Spreadsheet.RealTimeData interface.

Remarks

 **Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E5204>.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- Spreadsheet for WinForms: Real Time Data (RTD) function
- Spreadsheet for WPF: Real Time Data (RTD) function

Sheets Property

Provides access to a collection of all sheets contained in the [workbook](#).

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Sheets As SheetCollection
```

C#

```
public SheetCollection Sheets { get; }
```

Property Value

A DevExpress.Spreadsheet.SheetCollection object specifying a sheet collection.

Remarks

The DevExpress.Spreadsheet.SheetCollection collection returned by the **Sheets** property stores both worksheets (DevExpress.Spreadsheet.Worksheet) and chart sheets (DevExpress.Spreadsheet.ChartSheet) contained in a workbook.

The DevExpress.Spreadsheet.SheetCollection collection is useful when you need to access a sheet of any type. If you work with sheets of only one type, consider using one of the following collections:

- DevExpress.Spreadsheet.WorksheetCollection - returns a collection of worksheets contained in the workbook;
- DevExpress.Spreadsheet.ChartSheetCollection - returns a collection of chart sheets contained in the workbook.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

DevExpress.Spreadsheet.WorksheetCollection

DevExpress.Spreadsheet.ChartSheetCollection

Styles Property

Provides access to the collection of cell styles in the current workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Styles As StyleCollection
```

C#

```
public StyleCollection Styles { get; }
```

Property Value

A DevExpress.Spreadsheet.StyleCollection collection containing cell styles.

Remarks

By default, the DevExpress.Spreadsheet.StyleCollection contains Microsoft® Excel® built-in styles, including the *Normal* style that is applied to cells by default (DevExpress.Spreadsheet.StyleCollection.DefaultStyle). You can create custom styles, as well as copy, modify or delete existing styles (see the [How to: Create or Modify a Style](#) example).

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to format a cell, a range of cells, an entire row or an entire column by applying a [style](#).

1. Access the DevExpress.Spreadsheet.Style object that specifies the style to be applied to a cell or a range of cells. This style should be added to the [Styles](#) collection.
2. Assign the required style object to the DevExpress.Spreadsheet.Range.Style property of the cell, cell range, row or column object.

C#

(FormattingActions.cs)
Worksheet worksheet = workbook.Worksheets[0];
// Access the built-in "Good" MS Excel style from the Styles collection of the workbook.
Style styleGood = workbook.Styles[BuiltInStyleId.Good];
// Apply the "Good" style to a range of cells.
worksheet.Range["A1:C4"].Style = styleGood;
// Access a custom style that has been previously created in the loaded document by its name.
Style customStyle = workbook.Styles["Custom Style"];
// Apply the custom style to the cell.
worksheet.Cells["D6"].Style = customStyle;
// Apply the "Good" style to the eighth row.
worksheet.Rows[7].Style = styleGood;
// Apply the custom style to the "H" column.
worksheet.Columns["H"].Style = customStyle;

Visual Basic

```
(FormattingActions.vb)
Dim worksheet As Worksheet = workbook.Worksheets(0)
' Access the built-in "Good" MS Excel style from the Styles collection of the workbook.
Dim styleGood As Style = workbook.Styles(BuiltInStyleId.Good)
' Apply the "Good" style to a range of cells.
worksheet.Range("A1:C4").Style = styleGood
' Access a custom style that has been previously created in the loaded document by its name.
Dim customStyle As Style = workbook.Styles("Custom Style")
' Apply the custom style to the cell.
worksheet.Cells("D6").Style = customStyle
' Apply the "Good" style to the eighth row.
worksheet.Rows(7).Style = styleGood
' Apply the custom style to the "H" column.
worksheet.Columns("H").Style = customStyle
```

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[How to: Apply a Style to a Cell or Range of Cells](#)

[How to: Create or Modify a Style](#)

TableStyles Property

Provides access to the collection of styles to format tables in the workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property TableStyles As TableStyleCollection
```

C#

```
public TableStyleCollection TableStyles { get; }
```

Property Value

A DevExpress.Spreadsheet.TableStyleCollection object containing table styles.

Remarks

The **TableStyles** property returns the collection of styles that you can apply to tables to change their appearance. An individual table style is specified by the DevExpress.Spreadsheet.TableStyle object. To apply a style to a table, assign the required table style object to the DevExpress.Spreadsheet.Table.Style property.

By default, a workbook's collection of table styles contains Microsoft® Excel® built-in table styles that can be obtained by their ids (members of the BuiltInTableStyleId enumeration). Built-in table styles cannot be deleted or modified. However, you can create your own custom table styles, as well as duplicate, modify or delete them.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Tag Property

Gets or sets the data associated with a [Workbook](#) object.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Tag As Object
```

C#

```
public object Tag { get; set; }
```

Property Value

A [System.Object](#) that contains arbitrary data for a workbook. The default is **null**.

Remarks

Use the **Tag** property to store data associated with a workbook. Any type derived from the [System.Object](#) class can be assigned to this property.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Unit Property

Gets or sets a [unit of measure](#) used in the workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Unit As DocumentUnit
```

C#

```
public DocumentUnit Unit { get; set; }
```

Property Value

The DevExpress.Office.DocumentUnit enumeration member.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[UnitChanging](#)

[UnitChanged](#)

[Measure Units](#)

Worksheets Property

Gets a collection of [worksheets](#) contained in the [workbook](#).

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public ReadOnly Property Worksheets As WorksheetCollection
```

C#

```
public WorksheetCollection Worksheets { get; }
```

Property Value

A DevExpress.Spreadsheet.WorksheetCollection object that is a collection of worksheets.

Remarks

A workbook consists of one or more worksheets that are stored within the DevExpress.Spreadsheet.WorksheetCollection collection returned by the **Worksheets** property. Use members of the DevExpress.Spreadsheet.WorksheetCollection object to manage a workbook's set of worksheets. For example, you can access an individual worksheet by its name or index, [add a new worksheet](#), [remove an existing worksheet](#), [assign an active worksheet](#), etc.

For more examples on how to manage worksheets, see the [Worksheets](#) section.

Example

This example demonstrates how to access worksheets in a workbook. Use the [Worksheets](#) property to get a collection of worksheets contained in a workbook (the DevExpress.Spreadsheet.WorksheetCollection object). To get an individual worksheet by its index or name, use the DevExpress.Spreadsheet.WorksheetCollection.Item property.

C#

```
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
// Access a collection of worksheets.
WorksheetCollection worksheets = workbook.Worksheets;
// Access a worksheet by its index.
Worksheet worksheet1 = workbook.Worksheets[0];
// Access a worksheet by its name.
Worksheet worksheet2 = workbook.Worksheets["Sheet2"];
```

Visual Basic

```
Imports DevExpress.Spreadsheet
' ...
Dim workbook As New Workbook()
' Access a collection of worksheets.
Dim worksheets As WorksheetCollection = workbook.Worksheets
' Access a worksheet by its index.
Dim worksheet1 As Worksheet = workbook.Worksheets(0)
' Access a worksheet by its name.
Dim worksheet2 As Worksheet = workbook.Worksheets("Sheet2")
```

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [Workbook](#)
- [Worksheets](#)

Workbook Events

A non-visual component providing complete spreadsheet functionality.

The following tables list the members exposed by the [Workbook](#) type.

Public Events

	Name	Description
	ActiveSheetChanged	Occurs after an active worksheet in a workbook has been changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ActiveSheetChanging	Occurs when an active worksheet in a workbook is about to be changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforeExport	Occurs before the document is saved (exported to a certain format). Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforeImport	Occurs before a document is loaded (imported from an external source). Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeforePrintSheet	Occurs before printing a workbook.
	ClipboardDataObtained	Occurs after data on the clipboard is obtained and recognized, but before data is actually pasted. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ClipboardDataPasted	Occurs after data has been pasted from the clipboard onto a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ClipboardDataPasting	Occurs before data is pasted into destination cells. Use of this event in production code requires a license to the DevExpress

		Office File API or the DevExpress Universal Subscription.
	ColumnsInserted	Occurs after new columns have been added to a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ColumnsInserting	Occurs before inserting new columns into a worksheet.
	ColumnsRemoved	Occurs after columns have been deleted from a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ColumnsRemoving	Occurs before deleting columns from a worksheet.
	CommentInserted	Occurs when a comment is inserted.
	CommentInserting	Occurs before inserting a comment.
	CommentRemoved	Occurs when a comment is deleted.
	CommentRemoving	Occurs before deleting a comment.
	ContentChanged	Occurs when the workbook's undo history changes. By default, undo history is not logged for the Workbook instance, so the event is not raised. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CopiedRangePasted	Occurs after the range content has been pasted into target cells. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CopiedRangePasting	Occurs before the range content is pasted into target cells. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CustomAssemblyLoading	Occurs before a custom assembly is loaded for use as the Entity Framework data source during mail merge and allows cancelling loading. Use of this event in production code requires a license to the DevExpress

		Office File API or the DevExpress Universal Subscription.
	DefinedNameConflictResolving	Occurs when the formula or sheet being moved or copied contains a defined name which already exists on the destination worksheet or workbook. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DocumentClosing	Occurs when a document that has not yet been saved is about to be closed. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DocumentLoaded	Occurs after a document is loaded. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	DocumentSaved	Occurs after a document has been saved. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	EmptyDocumentCreated	Occurs when a new document is created. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	EncryptedFileIntegrityCheckFailed	Raises when the encrypted file did not pass the data integrity verification.
	EncryptedFilePasswordRequest	Raises when the DevExpress.XtraSpreadsheet.Import.WorkbookImportOptions.Password property is not set or contains a wrong password.
	InitializeDocument	Occurs before a document is loaded. Handle this event to set initial document settings. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	InvalidFormatException	Fires when the supplied data could not be recognized as data in the assumed format for import.

		Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ModifiedChanged	Occurs when the value of the Modified property is changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	PanesFrozen	Occurs after a worksheet area has been frozen. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	PanesUnfrozen	Occurs after a frozen worksheet area has been unlocked. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RangeCopied	Occurs after the range content has been copied. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RangeCopying	Occurs before a cell range is copied in a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RowsInserted	Occurs after new rows have been added to a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RowsInserting	Occurs before inserting new rows into a worksheet.
	RowsRemoved	Occurs after rows have been deleted from a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	RowsRemoving	Occurs before deleting rows from a worksheet.

	SchemaChanged	Occurs when a worksheet or defined name is renamed, inserted or deleted. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ScrollPositionChanged	Occurs when the scroll position changes in a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	SelectionChanged	Fires in response to changing cell selection in a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ShapeInserted	Occurs after a drawing object has been inserted into a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ShapeRemoved	Occurs after a drawing object has been removed from a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ShapeRemoving	Occurs before a drawing object is removed from a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ShapesCopying	Occurs before a drawing object is copied in a worksheet. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	SheetInserted	Occurs after a new worksheet has been added to a workbook. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	SheetRemoved	Occurs after a worksheet has been removed from a workbook.

		Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	SheetRemoving	Occurs before a worksheet is removed from a workbook.
	SheetRenamed	Occurs after a worksheet has been renamed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	SheetRenaming	Occurs when a worksheet is about to be renamed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	UnitChanged	Fires after a unit of measurement used in the workbook is changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	UnitChanging	Fires before a unit of measurement used within the workbook is changed. Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ValidateCustomSqlQuery	Allows validation of the custom SQL query created using the Data Source Wizard.

See Also[Workbook Members](#)[DevExpress.Spreadsheet Namespace](#)

ActiveSheetChanged Event

Occurs after an active worksheet in a workbook has been changed.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ActiveSheetChanged As ActiveSheetChangedEventHandler
```

C#

```
public event ActiveSheetChangedEventHandler ActiveSheetChanged
```

Event Data

The event handler receives an argument of type `ActiveSheetChangedEventArgs` containing data related to this event.

Remarks

Handle the **ActiveSheetChanged** event to perform specific actions each time an active worksheet is changed via the Spreadsheet API (by using the `DevExpress.Spreadsheet.WorksheetCollection.ActiveWorksheet` property). Before an active worksheet is changed, the [ActiveSheetChanging](#) event is raised.

The **ActiveSheetChanged** event occurs when a worksheet is removed from the workbook using the Spreadsheet API. In this case, the **ActiveSheetChanged** fires first, the [SheetRemoved](#) event fires next, and the [ActiveSheetChanging](#) event does not fire.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)

ActiveSheetChanging Event

Occurs when an active worksheet in a workbook is about to be changed.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ActiveSheetChanging As ActiveSheetChangingEventHandler
```

C#

```
public event ActiveSheetChangingEventHandler ActiveSheetChanging
```

Event Data

The event handler receives an argument of type `ActiveSheetChangingEventArgs` containing data related to this event.

Remarks

The **ActiveSheetChanging** event allows you to perform any actions before an active worksheet is changed via the Spreadsheet API (by using the `DevExpress.Spreadsheet.WorksheetCollection.ActiveWorksheet` property). You can cancel changing the active worksheet by setting the event parameter's **Cancel** property to **true**.

After an active worksheet has been changed, the [ActiveSheetChanged](#) event is raised.

When a worksheet is removed from the workbook, the **ActiveSheetChanging** event does not occur, although the [ActiveSheetChanged](#) event is fired.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

BeforeExport Event

Occurs before the document is saved (exported to a certain format).

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public BeforeExport As BeforeExportEventHandler
```

C#

```
public event BeforeExportEventHandler BeforeExport
```

Event Data

The event handler receives an argument of type SpreadsheetBeforeEventArgs containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[SaveDocument](#)

[How to: Save a Document to a File](#)

[How to: Export a Workbook to PDF](#)

BeforeImport Event

Occurs before a document is loaded (imported from an external source).

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public BeforeImport As BeforeImportEventHandler
```

C#

```
public event BeforeImportEventHandler BeforeImport
```

Event Data

The event handler receives an argument of type SpreadsheetBeforeImportEventArgs containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[LoadDocument](#)

[How to: Load a Document to a Workbook](#)

BeforePrintSheet Event

Occurs before printing a workbook.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public BeforePrintSheet As BeforePrintSheetEventHandler
```

C#

```
public event BeforePrintSheetEventHandler BeforePrintSheet
```

Event Data

The event handler receives an argument of type `BeforePrintSheetEventArgs` containing data related to this event.

Remarks

The **BeforePrintSheet** event fires for each worksheet being printed or exported to PDF. To cancel the print/export operation, set **e.Cancel** to **true**.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

ClipboardDataObtained Event

Occurs after data on the clipboard is obtained and recognized, but before data is actually pasted.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ClipboardDataObtained As ClipboardDataObtainedEventHandler
```

C#

```
public event ClipboardDataObtainedEventHandler ClipboardDataObtained
```

Event Data

The event handler receives an argument of type `ClipboardDataObtainedEventArgs` containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

ClipboardDataPasted Event

Occurs after data has been pasted from the clipboard onto a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ClipboardDataPasted As ClipboardDataPastedEventHandler
```

C#

```
public event ClipboardDataPastedEventHandler ClipboardDataPasted
```

Event Data

The event handler receives an argument of type `ClipboardDataPastedEventArgs` containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

ClipboardDataPasting Event

Occurs before data is pasted into destination cells.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ClipboardDataPasting As ClipboardDataPastingEventHandler
```

C#

```
public event ClipboardDataPastingEventHandler ClipboardDataPasting
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

ColumnsInserted Event

Occurs after new columns have been added to a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ColumnsInserted As ColumnsInsertedEventHandler
```

C#

```
public event ColumnsInsertedEventHandler ColumnsInserted
```

Event Data

The event handler receives an argument of type ColumnsChangedEventArgs containing data related to this event.

Remarks

Handle the **ColumnsInserted** event to perform specific actions each time new rows are inserted to a worksheet via the DevExpress.Spreadsheet.Column.Insert, DevExpress.Spreadsheet.ColumnCollection.Insert or the DevExpress.Spreadsheet.Worksheet.InsertCells methods.

Note

This event does not occur by default. The event fires only if the DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI property is **true**.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)

ColumnsInserting Event

Occurs before inserting new columns into a worksheet.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ColumnsInserting As ColumnsChangingEventHandler
```

C#

```
public event ColumnsChangingEventHandler ColumnsInserting
```

Event Data

The event handler receives an argument of type ColumnsChangingEventArgs containing data related to this event.

Remarks

The **ColumnsInserting** event allows you to perform any actions before new columns are inserted into a worksheet. You can cancel inserting new columns by setting the event parameter's **Cancel** property to **true**.

After new columns have been inserted, the [ColumnsInserted](#) event is raised.

Note

By default, the **ColumnsInserting** event does not occur when adding new columns in code. However, this event will also be triggered by changes made via an API if you set the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property (accessible via the **Workbook.Options.Events.RaiseOnModificationsViaAPI** notation) to **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [ColumnsInserted](#)

ColumnsRemoved Event

Occurs after columns have been deleted from a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ColumnsRemoved As ColumnsRemovedEventHandler
```

C#

```
public event ColumnsRemovedEventHandler ColumnsRemoved
```

Event Data

The event handler receives an argument of type ColumnsChangedEventArgs containing data related to this event.

Remarks

Handle the **ColumnsRemoved** event to perform specific actions each time columns are removed from a worksheet via the Spreadsheet API - usually by using the DevExpress.Spreadsheet.Column.Delete, DevExpress.Spreadsheet.ColumnCollection.Remove or DevExpress.Spreadsheet.Worksheet.DeleteCells methods.

Note

This event does not occur by default. The event fires only if the DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

ColumnsRemoving Event

Occurs before deleting columns from a worksheet.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ColumnsRemoving As ColumnsChangingEventHandler
```

C#

```
public event ColumnsChangingEventHandler ColumnsRemoving
```

Event Data

The event handler receives an argument of type ColumnsChangingEventArgs containing data related to this event.

Remarks

The **ColumnsRemoving** event allows you to perform any actions before columns are deleted from a worksheet. To prevent columns from being deleted, set the event parameter's **Cancel** property to **true**.

After columns have been removed from a worksheet, the [ColumnsRemoved](#) event is raised.

 **Note**

By default, the **ColumnsRemoving** event does not occur when removing columns in code. However, this event will also be triggered by changes made via an API if you set the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property (accessible via the **Workbook.Options.Events.RaiseOnModificationsViaAPI** notation) to **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [ColumnsRemoved](#)

CommentInserted Event

Occurs when a comment is inserted.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public CommentInserted As CommentChangedEventHandler
```

C#

```
public event CommentChangedEventHandler CommentInserted
```

Event Data

The event handler receives an argument of type `CommentChangedEventArgs` containing data related to this event.

Remarks

The **CommentInserted** event provides you with an index in the `DevExpress.Spreadsheet.Worksheet.Comments` collection when the end-user inserts a comment.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CommentInserting Event

Occurs before inserting a comment.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public CommentInserting As CommentInsertingEventHandler
```

C#

```
public event CommentInsertingEventHandler CommentInserting
```

Event Data

The event handler receives an argument of type `CommentInsertingEventArgs` containing data related to this event.

Remarks

The **CommentInserting** event provides you with information on a comment which the end-user is inserting and allows you to cancel insertion by setting **e.Cancel** to **true**.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CommentRemoved Event

Occurs when a comment is deleted.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public CommentRemoved As CommentChangedEventHandler
```

C#

```
public event CommentChangedEventHandler CommentRemoved
```

Event Data

The event handler receives an argument of type `CommentChangedEventArgs` containing data related to this event.

Remarks

The **CommentRemoved** event provides you with the deleted comment index in the `DevExpress.Spreadsheet.Worksheet.Comments` collection when the end-user removes a comment.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CommentRemoving Event

Occurs before deleting a comment.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public CommentRemoving As CommentRemovingEventHandler
```

C#

```
public event CommentRemovingEventHandler CommentRemoving
```

Event Data

The event handler receives an argument of type `CommentRemovingEventArgs` containing data related to this event.

Remarks

The **CommentRemoving** event provides you with an index in the `DevExpress.Spreadsheet.Worksheet.Comments` collection and allows you to cancel deletion.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

ContentChanged Event

Occurs when the workbook's undo history changes. By default, undo history is not logged for the Workbook instance, so the event is not raised.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ContentChanged As EventHandler
```

C#

```
public event EventHandler ContentChanged
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CopiedRangePasted Event

Occurs after the range content has been pasted into target cells.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public CopiedRangePasted As CopiedRangePastedEventHandler
```

C#

```
public event CopiedRangePastedEventHandler CopiedRangePasted
```

Event Data

The event handler receives an argument of type `CopiedRangePastedEventArgs` containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CopiedRangePasting Event

Occurs before the range content is pasted into target cells.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public CopiedRangePasting As CopiedRangePastingEventHandler
```

C#

```
public event CopiedRangePastingEventHandler CopiedRangePasting
```

Event Data

The event handler receives an argument of type `CopiedRangePastingEventArgs` containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CustomAssemblyLoading Event

Occurs before a custom assembly is loaded for use as the Entity Framework data source during mail merge and allows cancelling loading.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public CustomAssemblyLoading As SpreadsheetCustomAssemblyLoadingEventHandler
```

C#

```
public event SpreadsheetCustomAssemblyLoadingEventHandler CustomAssemblyLoading
```

Event Data

The event handler receives an argument of type SpreadsheetCustomAssemblyLoadingEventArgs containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

DefinedNameConflictResolving Event

Occurs when the formula or sheet being moved or copied contains a defined name which already exists on the destination worksheet or workbook.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public DefinedNameConflictResolving As DefinedNameConflictResolvingEventHandler
```

C#

```
public event DefinedNameConflictResolvingEventHandler DefinedNameConflictResolving
```

Event Data

The event handler receives an argument of type `DefinedNameConflictResolvingEventArgs` containing data related to this event.

Remarks

Note that when you're working with a [Workbook](#), you don't need to subscribe to the **DefinedNameConflictResolving** event, since a valid version of the copied name is automatically generated and used by default in the conflict situation. To change this name, handle the **DefinedNameConflictResolving** event and assign the necessary name to the `DevExpress.Spreadsheet.DefinedNameConflictResolvingEventArgs.NewName` parameter.

To use the defined name specified on the destination worksheet or workbook if the conflict occurs during the copy operation, set the `DevExpress.Spreadsheet.DefinedNameConflictResolvingEventArgs.UseExistingName` parameter to **true**.

The `DevExpress.Spreadsheet.DefinedNameConflictResolvingEventArgs.Validator` parameter provides access to the default validator that enables you to validate a new version of the conflict name before pasting it into the destination worksheet/workbook. You can also specify a custom validator by creating an object implementing the `DevExpress.XtraSpreadsheet.Services.IDefinedNameValidator` interface and assigning it to the `DevExpress.Spreadsheet.DefinedNameConflictResolvingEventArgs.Validator` property.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

DocumentClosing Event

Occurs when a document that has not yet been saved is about to be closed.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public DocumentClosing As CancelEventHandler
```

C#

```
public event CancelEventHandler DocumentClosing
```

Event Data


The event handler receives an argument of type [CancelEventArgs](#) containing data related to this event.

The following **CancelEventArgs** properties provide information specific to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.

Remarks

Handle the **DocumentClosing** event to specify the actions to perform before the document is closed. To cancel closing, set the **e.Cancel** value to **true**.

 **Note**

The **DocumentClosing** event is raised only if the document has been modified.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)

DocumentLoaded Event

Occurs after a document is loaded.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public DocumentLoaded As EventHandler
```

C#

```
public event EventHandler DocumentLoaded
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

The **DocumentLoaded** event fires after the [LoadDocument](#) method is executed when the document model is built and the loaded document is valid. Handle the **DocumentLoaded** event to ensure that the document is loaded completely so you can safely modify it.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[How to: Load a Document to a Workbook](#)

DocumentSaved Event

Occurs after a document has been saved.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public DocumentSaved As EventHandler
```

C#

```
public event EventHandler DocumentSaved
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

The **DocumentSaved** event fires after a document has been saved by an end-user at runtime or in code using the [SaveDocument](#) method.

The [BeforeExport](#) event occurs before a document is saved and allows you to specify export options related to the selected format.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[SaveDocument](#)

[How to: Save a Document to a File](#)

EmptyDocumentCreated Event

Occurs when a new document is created.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public EmptyDocumentCreated As EventHandler
```

C#

```
public event EventHandler EmptyDocumentCreated
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

The **EmptyDocumentCreated** event is fired after the [CreateNewDocument](#) method is executed.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[How to: Create a New Workbook](#)

EncryptedFileIntegrityCheckFailed Event

Raises when the encrypted file did not pass the data integrity verification.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public EncryptedFileIntegrityCheckFailed As EncryptedFileIntegrityCheckFailedEventHandler
```

C#

```
public event EncryptedFileIntegrityCheckFailedEventHandler EncryptedFileIntegrityCheckFailed
```

Event Data

The event handler receives an argument of type `EncryptedFileIntegrityCheckFailedEventArgs` containing data related to this event.

Remarks

SpreadsheetControl uses the Hash-based message authentication code (HMAC) to verify the document data integrity. If the document did not pass the code verification, the **EncryptedFileIntegrityCheckFailed** event is raised.

Note

The HMAC is calculated only for certain encryption types. When you save a password encrypted workbook, set the `DevExpress.Spreadsheet.EncryptionOptions.Type` to the `DevExpress.Spreadsheet.EncryptionType.Strong` to calculate HMAC.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

EncryptedFilePasswordRequest Event

Raises when the `DevExpress.XtraSpreadsheet.Import.WorkbookImportOptions.Password` property is not set or contains a wrong password.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public EncryptedFilePasswordRequest As EncryptedFilePasswordRequestEventHandler
```

C#

```
public event EncryptedFilePasswordRequestEventHandler EncryptedFilePasswordRequest
```

Event Data

The event handler receives an argument of type `EncryptedFilePasswordRequestEventArgs` containing data related to this event.

Remarks

The **EncryptedFilePasswordRequest** event allows to specify a password in code using the `DevExpress.Spreadsheet.EncryptedFilePasswordRequestEventArgs.Password` property.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

InitializeDocument Event

Occurs before a document is loaded. Handle this event to set initial document settings.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public InitializeDocument As EventHandler
```

C#

```
public event EventHandler InitializeDocument
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

InvalidFormatException Event

Fires when the supplied data could not be recognized as data in the assumed format for import.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public InvalidFormatException As InvalidFormatExceptionEventHandler
```

C#

```
public event InvalidFormatExceptionEventHandler InvalidFormatException
```

Event Data

The event handler receives an argument of type SpreadsheetInvalidFormatExceptionEventArgs containing data related to this event.

Remarks

If the data for import is in an incorrect format, the spreadsheet component silently cancels import if the **InvalidFormatException** is not handled. If you subscribe to the **InvalidFormatException** event, the event is fired when the component attempts to load incorrect data, so you can provide custom logic in this situation.

To throw an exception on loading an invalid document, set the DevExpress.XtraSpreadsheet.Import.WorkbookImportOptions.ThrowExceptionOnInvalidDocument property to **true** (import options are accessible via the **Workbook.Options.Import** notation).

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[How to: Save a Document to a File](#)

ModifiedChanged Event

Occurs when the value of the [Modified](#) property is changed.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ModifiedChanged As EventHandler
```

C#

```
public event EventHandler ModifiedChanged
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

PanesFrozen Event

Occurs after a worksheet area has been frozen.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public PanesFrozen As PanesFrozenEventHandler
```

C#

```
public event PanesFrozenEventHandler PanesFrozen
```

Event Data

The event handler receives an argument of type PanesFrozenEventArgs containing data related to this event.

Remarks

Handle the **PanesFrozen** event to perform specific actions when worksheet rows and columns are locked via the `DevExpress.Spreadsheet.Worksheet.FreezeRows`, `DevExpress.Spreadsheet.Worksheet.FreezeColumns`, or `DevExpress.Spreadsheet.Worksheet.FreezePanes` method.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [PanesUnfrozen](#)

PanesUnfrozen Event

Occurs after a frozen worksheet area has been unlocked.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public PanesUnfrozen As PanesUnfrozenEventHandler
```

C#

```
public event PanesUnfrozenEventHandler PanesUnfrozen
```

Event Data

The event handler receives an argument of type PanesUnfrozenEventArgs containing data related to this event.

Remarks

Handle the **PanesUnfrozen** event to perform specific actions when frozen rows and columns on a worksheet are unlocked via the DevExpress.Spreadsheet.Worksheet.UnfreezePanes method.

Note

This event does not occur by default. The event fires only if the DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [PanesFrozen](#)

RangeCopied Event

Occurs after the range content has been copied.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public RangeCopied As RangeCopiedEventHandler
```

C#

```
public event RangeCopiedEventHandler RangeCopied
```

Event Data

The event handler receives an argument of type RangeCopiedEventArgs containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

RangeCopying Event

Occurs before a cell range is copied in a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public RangeCopying As RangeCopyingEventHandler
```

C#

```
public event RangeCopyingEventHandler RangeCopying
```

Event Data

The event handler receives an argument of type RangeCopyingEventArgs containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

RowsInserted Event

Occurs after new rows have been added to a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public RowsInserted As RowsInsertedEventHandler
```

C#

```
public event RowsInsertedEventHandler RowsInserted
```

Event Data

The event handler receives an argument of type `RowsChangedEventArgs` containing data related to this event.

Remarks

Handle the **RowsInserted** event to perform specific actions each time new rows are inserted to a worksheet via the `DevExpress.Spreadsheet.Row.Insert`, `DevExpress.Spreadsheet.RowCollection.Insert` or `DevExpress.Spreadsheet.Worksheet.InsertCells` methods.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

RowsInserting Event

Occurs before inserting new rows into a worksheet.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public RowsInserting As RowsChangingEventHandler
```

C#

```
public event RowsChangingEventHandler RowsInserting
```

Event Data

The event handler receives an argument of type `RowsChangingEventArgs` containing data related to this event.

Remarks

The **RowsInserting** event allows you to perform any actions before adding new rows to a worksheet. You can cancel inserting new rows by setting the event parameter's **Cancel** property to **true**.

After new rows have been inserted, the [RowsInserted](#) event is raised.

Note

By default, the **RowsInserting** event does not occur when adding new rows in code. However, this event will also be triggered by changes made via an API if you set the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property (accessible via the **Workbook.Options.Events.RaiseOnModificationsViaAPI** notation) to **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [RowsInserted](#)

RowsRemoved Event

Occurs after rows have been deleted from a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public RowsRemoved As RowsRemovedEventHandler
```

C#

```
public event RowsRemovedEventHandler RowsRemoved
```

Event Data

The event handler receives an argument of type `RowsChangedEventArgs` containing data related to this event.

Remarks

Handle the **RowsRemoved** event to perform specific actions each time an row is removed from a worksheet using the Spreadsheet API methods such as the `DevExpress.Spreadsheet.Row.Delete`, `DevExpress.Spreadsheet.RowCollection.Remove` or `DevExpress.Spreadsheet.Worksheet.DeleteCells` methods.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

RowsRemoving Event

Occurs before deleting rows from a worksheet.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public RowsRemoving As RowsChangingEventHandler
```

C#

```
public event RowsChangingEventHandler RowsRemoving
```

Event Data

The event handler receives an argument of type `RowsChangingEventArgs` containing data related to this event.

Remarks

The **RowsRemoving** event allows you to perform any actions before removing rows from a worksheet. To prevent rows from being deleted, set the event parameter's **Cancel** property to **true**.

After rows have been removed from a worksheet, the [RowsRemoved](#) event is raised.

Note

By default, the **RowsRemoving** event does not occur when removing rows in code. However, this event will also be triggered by changes made via an API if you set the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property (accessible via the **Workbook.Options.Events.RaiseOnModificationsViaAPI** notation) to **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [RowsRemoved](#)

SchemaChanged Event

Occurs when a worksheet or defined name is renamed, inserted or deleted.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SchemaChanged As EventHandler
```

C#

```
public event EventHandler SchemaChanged
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

ScrollPositionChanged Event

Occurs when the scroll position changes in a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ScrollPositionChanged As ScrollPositionChangedEventHandler
```

C#

```
public event ScrollPositionChangedEventHandler ScrollPositionChanged
```

Event Data

The event handler receives an argument of type `ScrollPositionChangedEventArgs` containing data related to this event.

Remarks

- The **ScrollPositionChanged** event occurs in the following cases:
- When a worksheet is scrolled in code using the `DevExpress.Spreadsheet.Worksheet.ScrollTo`, `DevExpress.Spreadsheet.Worksheet.ScrollToColumn` or `DevExpress.Spreadsheet.Worksheet.ScrollToRow` method.
 - When the `DevExpress.Spreadsheet.Worksheet.FreezePanels`, `DevExpress.Spreadsheet.Worksheet.FreezeColumns`, `DevExpress.Spreadsheet.Worksheet.FreezeRows` or `DevExpress.Spreadsheet.Worksheet.UnfreezePanels` method is called to create or unlock frozen panels on a worksheet.

Handle this event to determine the name of the worksheet being scrolled and obtain the column and row indexes of the top-left cell of the currently visible area. If a worksheet contains frozen rows and columns, the **ColumnIndex** and **RowIndex** properties of the `DevExpress.Spreadsheet.ScrollPositionChangedEventArgs` class return indexes of the top-left cell of the currently visible *scrollable* area.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

SelectionChanged Event

Fires in response to changing cell selection in a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SelectionChanged As EventHandler
```

C#


```
public event EventHandler SelectionChanged
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

Handle the **SelectionChanged** event to perform specific actions each time another cell in a worksheet is selected. To change a cell that is currently selected, use the `DevExpress.Spreadsheet.Worksheet.SelectedCell` property.

 **Note**

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

ShapeInserted Event

Occurs after a [drawing object](#) has been inserted into a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ShapeInserted As ShapeChangedEventHandler
```

C#

```
public event ShapeChangedEventHandler ShapeInserted
```

Event Data

The event handler receives an argument of type ShapeChangedEventArgs containing data related to this event.

Remarks

Handle the **ShapeInserted** event to perform specific actions each time a drawing object is inserted into a worksheet. To obtain the type of the inserted drawing object, use the **ShapeType** parameter.

After a drawing object has been deleted from a worksheet, the [ShapeRemoved](#) event is raised.

Note

By default, the **ShapeInserted** event does not occur when inserting a new drawing object in code. However, this event will also be triggered by changes made via an API if you set the DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI property (accessible via the **Workbook.Options.Events.RaiseOnModificationsViaAPI** notation) to **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [ShapeRemoved](#)

ShapeRemoved Event

Occurs after a [drawing object](#) has been removed from a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ShapeRemoved As ShapeChangedEventHandler
```

C#

```
public event ShapeChangedEventHandler ShapeRemoved
```

Event Data

The event handler receives an argument of type ShapeChangedEventArgs containing data related to this event.

Remarks

Handle the **ShapeRemoved** event to perform specific actions each time a drawing object is removed from a worksheet. To obtain the type of the removed drawing object, use the **ShapeType** parameter.

To prevent a drawing object from being removed, use the [ShapeRemoving](#) event that is raised before a drawing object is deleted.

Note

By default, the **ShapeRemoved** event does not occur when removing a drawing object in code. However, this event will also be triggered by changes made via an API if you set the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property (accessible via the **Workbook.Options.Events.RaiseOnModificationsViaAPI** notation) to **true**.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)

ShapeRemoving Event

Occurs before a [drawing object](#) is removed from a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ShapeRemoving As ShapeChangingEventHandler
```

C#

```
public event ShapeChangingEventHandler ShapeRemoving
```

Event Data

The event handler receives an argument of type ShapeChangingEventArgs containing data related to this event.

Remarks

The **ShapeRemoving** event allows you to perform any actions before a drawing object is removed from the worksheet. To obtain the type of the drawing object for which the event has been raised, use the event parameter's **ShapeType** property. You can prevent a drawing object from being removed by setting the **Cancel** property to **true**.

After a drawing object has been removed, the [ShapeRemoved](#) event is raised.

Note

By default, the **ShapeRemoving** event does not occur. To trigger it, set the DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI property (accessible via the **Workbook.Options.Events.RaiseOnModificationsViaAPI** notation) to **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [ShapeRemoved](#)

ShapesCopying Event

Occurs before a drawing object is copied in a worksheet.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ShapesCopying As ShapesCopyingEventHandler
```

C#

```
public event ShapesCopyingEventHandler ShapesCopying
```

Event Data

The event handler receives an argument of type `ShapesCopyingEventArgs` containing data related to this event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

SheetInserted Event

Occurs after a new worksheet has been added to a workbook.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SheetInserted As SheetInsertedEventHandler
```

C#

```
public event SheetInsertedEventHandler SheetInserted
```

Event Data

The event handler receives an argument of type `SheetInsertedEventArgs` containing data related to this event.

Remarks

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

SheetRemoved Event

Occurs after a worksheet has been removed from a workbook.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SheetRemoved As SheetRemovedEventHandler
```

C#

```
public event SheetRemovedEventHandler SheetRemoved
```

Event Data

The event handler receives an argument of type `SheetRemovedEventArgs` containing data related to this event.

Remarks

Handle the **SheetRemoved** event to perform specific actions each time a worksheet is deleted using the `DevExpress.Spreadsheet.WorksheetCollection.Remove` or `DevExpress.Spreadsheet.WorksheetCollection.RemoveAt` methods.

When a worksheet is removed from the workbook, the [ActiveSheetChanged](#) event fires first, and the **SheetRemoved** event fires next. The [ActiveSheetChanging](#) event does not occur.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

SheetRemoving Event

Occurs before a worksheet is removed from a workbook.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SheetRemoving As SheetRemovingEventHandler
```

C#

```
public event SheetRemovingEventHandler SheetRemoving
```

Event Data

The event handler receives an argument of type `SheetRemovingEventArgs` containing data related to this event.

Remarks

The **SheetRemoving** event allows you to perform any actions before removing a worksheet from a workbook. You can cancel removing a worksheet by setting the event parameter's **Cancel** property to **true**.

After a worksheet has been removed from a workbook, the [SheetRemoved](#) event is raised.

Note

By default, the **SheetRemoving** event does not occur. To trigger it, set the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property (accessible via the **Workbook.Options.Events.RaiseOnModificationsViaAPI** notation) to **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [SheetRemoved](#)

SheetRenamed Event

Occurs after a worksheet has been renamed.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SheetRenamed As SheetRenamedEventHandler
```

C#

```
public event SheetRenamedEventHandler SheetRenamed
```

Event Data

The event handler receives an argument of type `SheetRenamedEventArgs` containing data related to this event.

Remarks

The **SheetRenamed** event fires after the `DevExpress.Spreadsheet.Worksheet.Name` property value has been changed by using the `DevExpress.Spreadsheet.Worksheet.Name` property. Handle this event to perform actions every time a worksheet is renamed.

To cancel renaming a worksheet, handle the [SheetRenaming](#) event that is raised before a worksheet is renamed.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

SheetRenaming Event

Occurs when a worksheet is about to be renamed.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public SheetRenaming As SheetRenamingEventHandler
```

C#

```
public event SheetRenamingEventHandler SheetRenaming
```

Event Data

The event handler receives an argument of type `SheetRenamingEventArgs` containing data related to this event.

Remarks

The **SheetRenaming** event allows you to perform an action before a worksheet is renamed using the `DevExpress.Spreadsheet.Worksheet.Name` property. You can cancel renaming a worksheet by setting the event parameter's **Cancel** property to **true**.

After a worksheet name has been changed, the [SheetRenamed](#) event is raised.

Note

This event does not occur by default. The event fires only if the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is **true**.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)

UnitChanged Event

Fires after a [unit of measurement](#) used in the workbook is changed.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public UnitChanged As EventHandler
```

C#

```
public event EventHandler UnitChanged
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

The **UnitChanged** event occurs after the [Unit](#) property is changed.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[UnitChanging](#)

[Measure Units](#)

UnitChanging Event

Fires before a [unit of measurement](#) used within the workbook is changed.

Use of this event in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public UnitChanging As EventHandler
```

C#

```
public event EventHandler UnitChanging
```

Event Data

The event handler receives an argument of type [EventArgs](#) containing data related to this event.

Remarks

Handle the **UnitChanging** event to validate or cancel the new setting of the document [measurement unit](#). This event occurs before the [Unit](#) property is changed.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[UnitChanged](#)

[Measure Units](#)

ValidateCustomSqlQuery Event

Allows validation of the custom SQL query created using the Data Source Wizard.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
event Public ValidateCustomSqlQuery As SpreadsheetValidateCustomSqlQueryEventHandler
```

C#

```
public event SpreadsheetValidateCustomSqlQueryEventHandler ValidateCustomSqlQuery
```

Event Data

The event handler receives an argument of type SpreadsheetValidateCustomSqlQueryEventArgs containing data related to this event.

Remarks

The **ValidateCustomSqlQuery** event occurs when the end-user enters custom SQL query text using the Data Source Wizard, if the DevExpress.XtraSpreadsheet.SpreadsheetDataSourceWizardOptions.EnableCustomSql property is set to **true**.

The DevExpress.Spreadsheet.SpreadsheetValidateCustomSqlQueryEventArgs.Sql property returns the text of SQL query. You can determine whether the text is valid, using custom criteria, and set the DevExpress.Spreadsheet.SpreadsheetValidateCustomSqlQueryEventArgs.Valid to **true** to proceed with the text or **false** to cancel and display a message specified by the DevExpress.Spreadsheet.SpreadsheetValidateCustomSqlQueryEventArgs.ExceptionMessage property.

Note

If the **ValidateCustomSqlQuery** event is not handled, a custom query used to obtain data from the SQL database should contain only SELECT statements (default validation criterion).

See Also

- [Workbook Class](#)
- [Workbook Members](#)
- [DevExpress.Spreadsheet Namespace](#)

Workbook Methods

A non-visual component providing complete spreadsheet functionality.

The following tables list the members exposed by the [Workbook](#) type.

Public Methods

	Name	Description
	AddService	Overloaded. Adds the specified service to the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	BeginUpdate	Locks the Workbook object until the EndUpdate or CancelUpdate method is called. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Calculate	Overloaded. Forces recalculation of the workbook. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CalculateFull	Forces a full calculation of the data in a workbook.
	CalculateFullRebuild	Forces a full calculation of the data and rebuilds the dependencies.
	CancelUpdate	Unlocks the Workbook object after it has been locked by the BeginUpdate method, without causing an immediate update. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CreateNewDocument	Creates and loads a new empty workbook. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Dispose	Releases resources associated with a Workbook instance. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

	EndUpdate	<p>Unlocks the Workbook object.</p> <p>Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	Equals	<p>Determines whether the specified System.Object instances are considered equal. (Inherited from Object)</p>
	Equals	<p>Determines whether the specified System.Object is equal to the current System.Object. (Inherited from Object)</p>
	Evaluate	<p>Overloaded. Evaluates the specified formula.</p> <p>Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ExportToHtml	<p>Overloaded. Exports the specified range to the specified stream in HTML format.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	ExportToPdf	<p>Overloaded. Exports the workbook to the specified stream in PDF format using the specified options.</p> <p>Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	GenerateMailMergeDocuments	<p>Performs a mail merge and returns the collection of resulting workbooks.</p> <p>Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	GetHashCode	<p>Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)</p>
	GetService	<p>Gets the service object of the specified type.</p> <p>Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
	GetService<T>	<p>Gets the specified service.</p>

		Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	LoadDocument	Overloaded. Loads the document from a System.Byte[] array.
	Print	Overloaded. Prints the document using the specified printer settings. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Protect	Protects the structure and windows of a workbook.
 	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	RemoveService	Overloaded. Removes the service of the specified type from the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ReplaceService<T>	Performs a service substitution. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	SaveDocument	Overloaded. Saves a document to an array of bytes in the specified format.
	Search	Overloaded. Performs a search in the current document using specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)
	Unprotect	Removes protection from a workbook.

See Also[Workbook Members](#)[DevExpress.Spreadsheet Namespace](#)

AddService Method

Adds the specified service to the service container.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
void AddService(Type serviceType, object serviceInstance)	Adds the specified service to the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void AddService(Type serviceType, ServiceCreatorCallback callback)	Adds the specified service to the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void AddService(Type serviceType, object serviceInstance, bool promote)	Adds the specified service to the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void AddService(Type serviceType, ServiceCreatorCallback callback, bool promote)	Adds the specified service to the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.AddService Overload List](#)

Adds the specified service to the service container.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AddService(  
    ByVal serviceType As Type,  
    ByVal serviceInstance As Object  
)
```

C#

```
public void AddService(  
    Type serviceType,  
    object serviceInstance  
)
```

Parameters

serviceType

The type of service to add.

serviceInstance

An instance of the service type to add. This object must implement or inherit from the type indicated by the `serviceType` parameter.

Remarks

The Workbook implements the `System.IServiceProvider` and `System.ComponentModel.Design.IServiceContainer` interfaces. In addition to providing services, it also provides a mechanism for adding and removing services. To obtain a service, call the [GetService](#) method.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.AddService Overload List](#)

[GetService](#)

Adds the specified service to the service container.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AddService(  
    ByVal serviceType As Type,  
    ByVal callback As ServiceCreatorCallback  
)
```

C#

```
public void AddService(  
    Type serviceType,  
    ServiceCreatorCallback callback  
)
```

Parameters

serviceType

The type of service to add.

callback

A callback object that is used to create the service. This allows a service to be declared as available, but delays the creation of the object until the service is requested.

Remarks

The Workbook implements the `System.IServiceProvider` and `System.ComponentModel.Design.IServiceContainer` interfaces. In addition to providing services, it also provides a mechanism for adding and removing services. To obtain a service, call the [GetService](#) method.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.AddService Overload List](#)

[GetService](#)

Adds the specified service to the service container.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub AddService(  
    ByVal serviceType As Type,  
    ByVal serviceInstance As Object,  
    ByVal promote As Boolean  
)
```

C#

```
public void AddService(  
    Type serviceType,  
    object serviceInstance,  
    bool promote  
)
```

Parameters

serviceType

The type of service to add.

serviceInstance

An instance of the service type to add. This object must implement or inherit from the type indicated by the serviceType parameter.

promote

true, to promote this request to any parent service containers; otherwise, **false**.

Remarks

The Workbook implements the System.IServiceProvider and System.ComponentModel.Design.IServiceContainer interfaces. In addition to providing services, it also provides a mechanism for adding and removing services. To obtain a service, call the [GetService](#) method.

When a service is added, it can be added with instructions to promote it. When a service is promoted, it is added to any parent service container, on up, until the top of the service container tree is reached.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.AddService Overload List](#)

[GetService](#)

Adds the specified service to the service container.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub AddService(  
    ByVal serviceType As Type,  
    ByVal callback As ServiceCreatorCallback,  
    ByVal promote As Boolean  
)
```

C#

```
public void AddService(  
    Type serviceType,  
    ServiceCreatorCallback callback,  
    bool promote  
)
```

Parameters

serviceType

The type of service to add.

callback

A callback object that is used to create the service. This allows a service to be declared as available, but delays the creation of the object until the service is requested.

promote

true, to promote this request to any parent service containers; otherwise, **false**.

Remarks

The Workbook implements the System.IServiceProvider and System.ComponentModel.Design.IServiceContainer interfaces. In addition to providing services, it also provides a mechanism for adding and removing services. To obtain a service, call the [GetService](#) method.

When a service is added, it can be added with instructions to promote it. When a service is promoted, it is added to any parent service container, on up, until the top of the service container tree is reached.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.AddService Overload List](#)

[GetService](#)

BeginUpdate Method

Locks the [Workbook](#) object until the [EndUpdate](#) or [CancelUpdate](#) method is called.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub BeginUpdate()
```

C#

```
public void BeginUpdate()
```

Remarks

The **BeginUpdate**, [EndUpdate](#) and [CancelUpdate](#) methods use an internal counter to implement the update functionality. The counter's initial value is 0. The BeginUpdate method increments the counter. EndUpdate and CancelUpdate decrement the counter. The updates are enabled if the counter's value is 0. Each call to BeginUpdate must be paired with a call to EndUpdate or CancelUpdate. If a call to BeginUpdate is made without a corresponding call to EndUpdate/CancelUpdate afterwards, or it's not called because an exception occurred, the object will no longer be updated. To ensure that EndUpdate/CancelUpdate is always called even if an exception occurs, use the **try...finally** statement.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[EndUpdate](#)

[CancelUpdate](#)

Calculate Method

Forces recalculation of the workbook.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
void Calculate()	Forces recalculation of the workbook. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void Calculate(Worksheet sheet)	Forces recalculation of the specified worksheet in a workbook.
void Calculate(Range range)	Forces recalculation of the specified cell range in a worksheet.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.Calculate Overload List](#)

Forces recalculation of the workbook.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Calculate()
```

C#

```
public void Calculate()
```

Remarks

When the **Calculate** is called, all workbook modifications are applied and all cells are calculated.

Recalculation of the entire workbook can take a significant amount of time and resources. To calculate only a formula contained in a single cell, use the [Evaluate](#) method.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.Calculate Overload List](#)

Forces recalculation of the specified worksheet in a workbook.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Calculate(  
    ByVal sheet As Worksheet  
)
```

C#

```
public void Calculate(  
    Worksheet sheet  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet object containing formulas to be recalculated.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.Calculate Overload List](#)

Calculation

[How to implement a service to manage the process of worksheet cell calculation](#)

Forces recalculation of the specified cell range in a worksheet.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub Calculate(  
    ByVal range As Range  
)
```

C#

```
public void Calculate(  
    Range range  
)
```

Parameters

range

A DevExpress.Spreadsheet.Range object containing formulas to be recalculated.

Remarks

The **Calculate** method calculates all cells contained in the specified range. If the DevExpress.Spreadsheet.CalculationOptions.Iterative option is enabled and the DevExpress.Spreadsheet.DocumentOptions.CalculationEngineType property is set to DevExpress.Spreadsheet.CalculationEngineType.ChainBased, the dependency tree is not constructed and the cells are calculated from left to right, top to bottom.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.Calculate Overload List](#)

Calculation

[How to implement a service to manage the process of worksheet cell calculation](#)

CalculateFull Method

Forces a full calculation of the data in a workbook.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub CalculateFull()
```

C#

```
public void CalculateFull()
```

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CalculateFullRebuild Method

Forces a full calculation of the data and rebuilds the dependencies.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub CalculateFullRebuild()
```

C#

```
public void CalculateFullRebuild()
```

Remarks

Reserved for future use. The **CalculateFullRebuild** method is implemented for compatibility with the Microsoft Excel API. Currently, the method is equivalent to the [CalculateFull](#) method.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

CancelUpdate Method

Unlocks the [Workbook](#) object after it has been locked by the **BeginUpdate** method, without causing an immediate update.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub CancelUpdate()
```

C#

```
public void CancelUpdate()
```

Remarks

See [BeginUpdate](#) to learn more.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[BeginUpdate](#)

[EndUpdate](#)

CreateNewDocument Method

Creates and loads a new empty workbook.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function CreateNewDocument() As Boolean
```

C#

```
public bool CreateNewDocument()
```

Return Value

true, if the document is created successfully; otherwise, **false**.

Remarks

If the `DevExpress.XtraSpreadsheet.WorkbookEventOptions.RaiseOnModificationsViaAPI` property is set to **true** and the document has unsaved changes (the [Modified](#) property is **true**), using the **CreateNewDocument** method will raise the [DocumentClosing](#) event.

By handling the [DocumentClosing](#) event, you can set **e.Cancel** to **true** to prevent a document from closing. In this case, the **CreateNewDocument** method will return **false**.

If the new document was created and successfully loaded, the **CreateNewDocument** method returns **true**.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

Dispose Method

Releases resources associated with a [Workbook](#) instance.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Dispose()
```

C#

```
public void Dispose()
```

Remarks

Use this method to release unmanaged resources held by an instance of the [Workbook](#) class. You are advised to call this method when a particular Workbook instance is no longer used.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

EndUpdate Method

Unlocks the [Workbook](#) object.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub EndUpdate()
```

C#

```
public void EndUpdate()
```

Remarks

See [BeginUpdate](#) to learn more.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[BeginUpdate](#)

[CancelUpdate](#)

Evaluate Method

Evaluates the specified formula.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
ParameterValue Evaluate(string formula)	Evaluates the specified formula. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
ParameterValue Evaluate(string formula, FormulaEvaluationContext context)	Evaluates the specified formula in a certain context. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.Evaluate Overload List](#)

Evaluates the specified formula.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Evaluate(  
    ByVal formula As String  
) As ParameterValue
```

C#

```
public ParameterValue Evaluate(  
    string formula  
)
```

Parameters

formula

A string that is the formula to be evaluated.

Return Value

A DevExpress.Spreadsheet.Functions.ParameterValue object that is the calculation result and may contain a value, a reference or a calculation error.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.Evaluate Overload List](#)

Evaluates the specified formula in a certain context.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Evaluate(  
    ByVal formula As String,  
    ByVal context As FormulaEvaluationContext  
) As ParameterValue
```

C#

```
public ParameterValue Evaluate(  
    string formula,  
    FormulaEvaluationContext context  
)
```

Parameters

formula

A string that is the formula to be evaluated.

context

A DevExpress.Spreadsheet.Functions.FormulaEvaluationContext object containing information on the row, column and the current culture settings.

Return Value

A DevExpress.Spreadsheet.Functions.ParameterValue object that is the calculation result and may contain a value, a reference or a calculation error.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.Evaluate Overload List](#)

ExportToHtml Method

Exports the specified range to the specified file in HTML format.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
void ExportToHtml(string fileName, Range range)	Exports the specified range to the specified file in HTML format. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToHtml(string fileName, int sheetIndex)	Exports the specified worksheet to the specified file in HTML format. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToHtml(string fileName, Worksheet sheet)	Exports the specified worksheet to the specified file in HTML format. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToHtml(string fileName, HtmlDocumentExporterOptions options)	Exports the document's data to the specified file in HTML format using the specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToHtml(Stream stream, int sheetIndex)	Exports the specified worksheet to the specified stream in HTML format. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToHtml(Stream stream, HtmlDocumentExporterOptions options)	Exports the document's data to the specified stream in HTML format using the specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToHtml(Stream stream, Worksheet sheet)	Exports the specified worksheet to the specified stream in HTML format. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToHtml(Stream stream, Range range)	Exports the specified range to the specified stream in HTML format. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.ExportToHtml Overload List](#)

Exports the specified range to the specified file in HTML format.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub ExportToHtml(  
    ByVal fileName As String,  
    ByVal range As Range  
)
```

C#

```
public void ExportToHtml(  
    string fileName,  
    Range range  
)
```

Parameters

fileName

A [System.String](#) value which contains the full path (including the file name and extension) specifying where the HTML file will be created.

range

A DevExpress.Spreadsheet.Range object to be exported to HTML.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.ExportToHtml Overload List](#)

Exports the specified worksheet to the specified file in HTML format.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub ExportToHtml(  
    ByVal fileName As String,  
    ByVal sheetIndex As Integer  
)
```

C#

```
public void ExportToHtml(  
    string fileName,  
    int sheetIndex  
)
```

Parameters

fileName

A [System.String](#) value which contains the full path (including the file name and extension) specifying where the HTML file will be created.

sheetIndex

An integer value that is the index of the worksheet to be exported to HTML.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.ExportToHtml Overload List](#)

Exports the specified worksheet to the specified file in HTML format.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToHtml(  
    ByVal fileName As String,  
    ByVal sheet As Worksheet  
)
```

C#

```
public void ExportToHtml(  
    string fileName,  
    Worksheet sheet  
)
```

Parameters

fileName

A [System.String](#) value which contains the full path (including the file name and extension) specifying where the HTML file will be created.

sheet

A DevExpress.Spreadsheet.Worksheet object to be exported to HTML.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.ExportToHtml Overload List](#)

Exports the document's data to the specified file in HTML format using the specified options.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToHtml(  
    ByVal fileName As String,  
    ByVal options As HtmlDocumentExporterOptions  
)
```

C#

```
public void ExportToHtml(  
    string fileName,  
    HtmlDocumentExporterOptions options  
)
```

Parameters

fileName

A [System.String](#) value which contains the full path (including the file name and extension) specifying where the HTML file will be created.

options

A DevExpress.XtraSpreadsheet.Export.HtmlDocumentExporterOptions instance containing required export options.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)

[Workbook.ExportToHtml Overload List](#)

Exports the specified worksheet to the specified stream in HTML format.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub ExportToHtml(  
    ByVal stream As Stream,  
    ByVal sheetIndex As Integer  
)
```

C#

```
public void ExportToHtml(  
    Stream stream,  
    int sheetIndex  
)
```

Parameters

stream

A [System.IO.Stream](#) object to which the created HTML file should be sent.

sheetIndex

An integer value that is the index of the worksheet to be exported to HTML.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.ExportToHtml Overload List](#)

Exports the document's data to the specified stream in HTML format using the specified options.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub ExportToHtml(  
    ByVal stream As Stream,  
    ByVal options As HtmlDocumentExporterOptions  
)
```

C#

```
public void ExportToHtml(  
    Stream stream,  
    HtmlDocumentExporterOptions options  
)
```

Parameters

stream

A [System.IO.Stream](#) object to which the created HTML file should be sent.

options

A DevExpress.XtraSpreadsheet.Export.HtmlDocumentExporterOptions instance containing required export options.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.ExportToHtml Overload List](#)

Exports the specified worksheet to the specified stream in HTML format.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToHtml(  
    ByVal stream As Stream,  
    ByVal sheet As Worksheet  
)
```

C#

```
public void ExportToHtml(  
    Stream stream,  
    Worksheet sheet  
)
```

Parameters

stream

A [System.IO.Stream](#) object to which the created HTML file should be sent.

sheet

A DevExpress.Spreadsheet.Worksheet object to be exported to HTML.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.ExportToHtml Overload List](#)

Exports the specified range to the specified stream in HTML format.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToHtml(  
    ByVal stream As Stream,  
    ByVal range As Range  
)
```

C#

```
public void ExportToHtml(  
    Stream stream,  
    Range range  
)
```

Parameters

stream

A [System.IO.Stream](#) object to which the created HTML file should be sent.

range

A DevExpress.Spreadsheet.Range object to be exported to HTML.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.ExportToHtml Overload List](#)

ExportToPdf Method

Exports the workbook to the specified file path in PDF format.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
void ExportToPdf(string fileName)	Exports the workbook to the specified file path in PDF format. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToPdf(Stream stream)	Exports the workbook to the specified stream in PDF format. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToPdf(string fileName, PdfExportOptions options)	Exports the workbook to the specified file path in PDF format using the specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void ExportToPdf(Stream stream, PdfExportOptions options)	Exports the workbook to the specified stream in PDF format using the specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.ExportToPdf Overload List](#)

Exports the workbook to the specified file path in PDF format.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToPdf(  
    ByVal fileName As String  
)
```

C#

```
public void ExportToPdf(  
    string fileName  
)
```

Parameters

fileName

A [System.String](#) value which specifies the file name (including the full path) for the created PDF file.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.ExportToPdf Overload List](#)
[How to: Export a Workbook to PDF](#)
[How to: Save a Document to a File](#)

Exports the workbook to the specified stream in PDF format.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub ExportToPdf(  
    ByVal stream As Stream  
)
```

C#

```
public void ExportToPdf(  
    Stream stream  
)
```

Parameters

stream

A [System.IO.Stream](#) object to which the created PDF file should be sent.

Example**Show Me**

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4339>.

This example demonstrates how to save a workbook in PDF format using the [ExportToPdf](#) method.

C#

```
(ExportActions.cs)  
using (FileStream pdfFileStream = new FileStream("Documents\\Document_PDF.pdf", FileMode.Create))  
{  
    workbook.ExportToPdf(pdfFileStream);  
}
```

Visual Basic

```
(ExportActions.vb)  
Using pdfFileStream As New FileStream("Documents\\Document_PDF.pdf", FileMode.Create)  
    workbook.ExportToPdf(pdfFileStream)  
End Using
```

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.ExportToPdf Overload List](#)
[How to: Save a Document to a File](#)

Exports the workbook to the specified file path in PDF format using the specified options.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToPdf(  
    ByVal fileName As String,  
    ByVal options As PdfExportOptions  
)
```

C#

```
public void ExportToPdf(  
    string fileName,  
    PdfExportOptions options  
)
```

Parameters

fileName

A [System.String](#) which specifies the file name (including the full path) for the created PDF file.

options

A DevExpress.XtraPrinting.PdfExportOptions object specifying export settings.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.ExportToPdf Overload List](#)

Exports the workbook to the specified stream in PDF format using the specified options.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub ExportToPdf(  
    ByVal stream As Stream,  
    ByVal options As PdfExportOptions  
)
```

C#

```
public void ExportToPdf(  
    Stream stream,  
    PdfExportOptions options  
)
```

Parameters

stream

A [System.IO.Stream](#) object to which the created PDF file should be sent.

options

A DevExpress.XtraPrinting.PdfExportOptions object specifying export settings.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.ExportToPdf Overload List](#)

GenerateMailMergeDocuments Method

Performs a mail merge and returns the collection of resulting workbooks.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GenerateMailMergeDocuments() As IList(of IWorkbook)
```

C#

```
public IList<IWorkbook> GenerateMailMergeDocuments()
```

Return Value

An IList object that contains workbooks generated after a mail merge is preformed.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

GetService Method

Gets the service object of the specified type.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetService(  
    ByVal serviceType As Type  
) As Object
```

C#

```
public object GetService(  
    Type serviceType  
)
```

Parameters

serviceType

An object that specifies the type of service object to get.

Return Value

A service object of the specified type, or a null reference (Nothing in Visual Basic) if there is no service object of this type.

Remarks

Use this method to enable your application objects to obtain a service of the Workbook in order to employ methods of the service. The Workbook implements System.IServiceProvider interface, so you can tell it what type of service you wish to retrieve with the **GetService** method; and if a service is available, it is offered to the caller object.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

GetService<T> Method

Gets the specified service.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetService(of T) () As T
```

C#

```
public T GetService<T>()
```

Return Value

A service object of the specified type or null for reference types and zero for numeric value types if a service is not available.

Remarks

Use this method to enable your application objects to obtain a service of the Workbook in order to employ methods of the service. The Workbook implements System.IServiceProvider interface, so you can tell it what type of service you wish to retrieve with the **GetService<T>** method; and if a service is available, it is offered to the caller object.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

LoadDocument Method

Loads a document from a file.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
bool LoadDocument(string fileName)	Loads a document from a file. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
bool LoadDocument(Byte[] buffer)	Loads the document from a System.Byte[] array.
bool LoadDocument(Stream stream)	Loads the document from a stream.
bool LoadDocument(Byte[] buffer, DocumentFormat format)	Loads a document from a byte array.
bool LoadDocument(string fileName, DocumentFormat format)	Loads a document from a file, specifying the document format. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
bool LoadDocument(Stream stream, DocumentFormat format)	Loads a document from a stream, specifying the document format. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also
[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.LoadDocument Overload List](#)

Loads a document from a file.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function LoadDocument(  
    ByVal fileName As String  
) As Boolean
```

C#

```
public bool LoadDocument(  
    string fileName  
)
```

Parameters

fileName
A string specifying the file to load (including the full path).

Return Value

true, if a document is loaded successfully; otherwise, **false**.

Remarks

When you use this method to load a document from a file, the document format is identified automatically based on its content (regardless of the filename extension).

If the data you are trying to load has an incorrect format, the [InvalidFormatException](#) event fires. You can subscribe to this event and perform required actions to resolve this situation. To throw an exception on loading an invalid document, set the `DevExpress.XtraSpreadsheet.Import.WorkbookImportOptions.ThrowExceptionOnInvalidDocument` property to **true** (import options are accessible via the **Workbook.Options.Import** notation).

To determine the moment you can safely modify a loaded document, handle the [DocumentLoaded](#) event.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.LoadDocument Overload List](#)

[SaveDocument](#)

[How to: Load a Document to a Workbook](#)

[How to: Save a Document to a File](#)

Loads the document from a `System.Byte[]` array.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function LoadDocument(  
    ByVal buffer As Byte()  
) As Boolean
```

C#

```
public bool LoadDocument(  
    Byte[] buffer  
)
```

Parameters

buffer

A `System.Byte[]` object that is an array of bytes containing document data.

Return Value

true, if the document is loaded successfully; otherwise, **false**.

Remarks

The format of the document loaded using this **LoadDocument** method overload is detected automatically by the built-in `DevExpress.XtraSpreadsheet.Services.IFormatDetectorService` implementation.

If the format detection fails, an `DevExpress.Spreadsheet.ISpreadsheetComponent.InvalidFormatException` is raised.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.LoadDocument Overload List](#)

Loads the document from a stream.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function LoadDocument(  
    ByVal stream As Stream  
) As Boolean
```

C#

```
public bool LoadDocument(  
    Stream stream  
)
```

Parameters

stream

An [System.IO.Stream](#) object that is the stream from which the document is loaded.

Return Value

true, if the document is loaded successfully; otherwise, **false**.

Remarks

The target stream can use a non-seekable stream to load a document. In this case, the stream buffers automatically.

The format of the document loaded from a stream is detected automatically by the built-in `DevExpress.XtraSpreadsheet.Services.IFormatDetectorService` service implementation. The following formats can be detected:

- XLSX, XLSM, XLTX, XLTM (non encrypted files only);
- XLS, XLT;
- CSV, TXT (only if loaded from a `System.IO.FileStream` instance).

If the format detection fails, an `DevExpress.Spreadsheet.ISpreadsheetComponent.InvalidFormatException` is raised.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.LoadDocument Overload List](#)

Loads a document from a byte array.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function LoadDocument(  
    ByVal buffer As Byte[],  
    ByVal format As DocumentFormat  
) As Boolean
```

C#

```
public bool LoadDocument(  
    Byte[] buffer,  
    DocumentFormat format  
)
```

Parameters

buffer

A `System.Byte[]` object that is an array of bytes containing document data in the specified format.

format

A `DevExpress.Spreadsheet.DocumentFormat` enumeration member specifying the format of the document to be loaded.

Return Value

true, if the document is successfully loaded; otherwise, **false**.

Remarks

The **LoadDocument** method can be used to load a workbook stored in an external database. Refer to the [How to: Store a Workbook in the Database](#) document for additional information.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.LoadDocument Overload List](#)
[SaveDocument](#)

Loads a document from a file, specifying the document format.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function LoadDocument(  
    ByVal fileName As String,  
    ByVal format As DocumentFormat  
) As Boolean
```

C#

```
public bool LoadDocument(  
    string fileName,  
    DocumentFormat format  
)
```

Parameters

fileName

A string specifying the file to load (including the full path).

format

One of the DevExpress.Spreadsheet.DocumentFormat members.

Return Value

true, if a document is loaded successfully; otherwise, **false**.

Remarks

If the data you are trying to load has an incorrect format, the [InvalidFormatException](#) event fires. You can subscribe to this event and perform required actions to resolve this situation. To throw an exception on loading an invalid document, set the `DevExpress.XtraSpreadsheet.Import.WorkbookImportOptions.ThrowExceptionOnInvalidDocument` property to **true** (import options are accessible via the **Workbook.Options.Import** notation).

To determine the moment you can safely modify a loaded document, handle the [DocumentLoaded](#) event.

Example

C#	
<pre>// Add a reference to the DevExpress.Docs.dll assembly. using DevExpress.Spreadsheet; // ... Workbook workbook = new Workbook(); // Load a workbook from the file. workbook.LoadDocument("Documents\\Document.xlsx", DocumentFormat.Xlsx);</pre>	

Visual Basic	
<pre>' Add a reference to the DevExpress.Docs.dll assembly. Imports DevExpress.Spreadsheet ' ... Private workbook As New Workbook() ' Load a workbook from the file. workbook.LoadDocument("Documents\\Document.xlsx", DocumentFormat.Xlsx)</pre>	

See Also

[Workbook Class](#)

[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.LoadDocument Overload List](#)
[SaveDocument](#)
[How to: Load a Document to a Workbook](#)
[How to: Save a Document to a File](#)

Loads a document from a stream, specifying the document format.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function LoadDocument(  
    ByVal stream As Stream,  
    ByVal format As DocumentFormat  
) As Boolean
```

C#

```
public bool LoadDocument(  
    Stream stream,  
    DocumentFormat format  
)
```

Parameters

stream

The stream from which to load a document.

format

One of the DevExpress.Spreadsheet.DocumentFormat members.

Return Value

true, if a document is loaded successfully; otherwise, **false**.

Remarks

If the data you are trying to load has an incorrect format, the [InvalidFormatException](#) event fires. You can subscribe to this event and perform actions required to resolve this situation. To throw an exception on loading an invalid document, set the DevExpress.XtraSpreadsheet.Import.WorkbookImportOptions.ThrowExceptionOnInvalidDocument property to **true** (import options are accessible via the **Workbook.Options.Import** notation).

To determine the moment you can safely modify a loaded document, handle the [DocumentLoaded](#) event.

The **LoadDocument** method can use a non-seekable stream to load a document. In this case, the stream buffers automatically.



Note

With the *format* method parameter passed as DevExpress.Spreadsheet.DocumentFormat.Undefined, the loaded document format is detected automatically.

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.LoadDocument Overload List](#)
[SaveDocument](#)
[How to: Load a Document to a Workbook](#)
[How to: Save a Document to a File](#)

Print Method

Prints the document to the default printer.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
void Print()	Prints the document to the default printer. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void Print(PrinterSettings printerSettings)	Prints the document using the specified printer settings. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also
[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.Print Overload List](#)

Prints the document to the default printer.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Sub Print()`
C#
`public void Print()`

Remarks

Use the current **Print** method overload to send the specified workbook to the default printer. To select another printer and change printer settings, create a System.Drawing.Printing.PrinterSettings class instance and pass it to the **Print** method as a parameter.

To print a specific sheet in a workbook, use the DevExpress.Spreadsheet.Sheet.Print method.

To define general page options, use properties of the DevExpress.Spreadsheet.WorksheetView object accessible from the DevExpress.Spreadsheet.Worksheet.ActiveView property. WorksheetView enables you to specify page orientation, margins and paper size settings.

The DevExpress.Spreadsheet.WorksheetPrintOptions object's properties allow you to specify more print-specific options, which include scaling, printing gridlines, titles, row and column headings, setting page order and more.

For more information on how to specify print settings and print a workbook, refer to the [Printing](#) example section.

See Also
[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.Print Overload List](#)
DevExpress.Spreadsheet.WorksheetView
DevExpress.Spreadsheet.WorksheetPrintOptions

Printing

Prints the document using the specified printer settings.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Print(  
    ByVal printerSettings As PrinterSettings  
)
```

C#

```
public void Print(  
    PrinterSettings printerSettings  
)
```

Parameters

printerSettings

A System.Drawing.Printing.PrinterSettings object that contains printer settings.

Remarks

Use the current **Print** method overload to print a workbook using the custom printer settings specified by the System.Drawing.Printing.PrinterSettings class instance. For example, the **PrinterSettings.Copies** property allows you to set the number of copies to print, the **PrinterSettings.PrinterName** property determines the printer to use, and the **PrinterSettings.PrintRange** property defines a print range. If **PrinterSettings.PrintRange** is **SomePages**, use the **PrinterSettings.FromPage** and **PrinterSettings.ToPage** properties to specify what pages should be printed.

C#

```
using DevExpress.Spreadsheet;  
using System.Drawing.Printing;  
// ...  
// Create a new Workbook object.  
Workbook workbook = new Workbook();  
// Load a document from a file.  
workbook.LoadDocument("Documents\\Document.xlsx");  
// Create an object containing printer settings.  
PrinterSettings printerSettings = new PrinterSettings();  
// Define the printer to use.  
printerSettings.PrinterName = "Microsoft Print to PDF";  
printerSettings.PrintToFile = true;  
printerSettings.PrintFileName = "Documents\\PrintedDocument.pdf";  
// Specify that the first three pages should be printed.  
printerSettings.PrintRange = PrintRange.SomePages;  
printerSettings.FromPage = 1;  
printerSettings.ToPage = 3;  
// Set the number of copies to print.  
printerSettings.Copies = 1;  
// Print the workbook using the specified printer settings.  
workbook.Print(printerSettings);
```

Visual Basic

```
Imports DevExpress.Spreadsheet
Imports System.Drawing.Printing
' ...
' Create a new Workbook object.
Private workbook As New Workbook()
' Load a document from a file.
workbook.LoadDocument("Documents\Document.xlsx")
' Create an object containing printer settings.
Dim printerSettings As New PrinterSettings()
' Define the printer to use.
printerSettings.PrinterName = "Microsoft Print to PDF"
printerSettings.PrintToFile = True
printerSettings.PrintFileName = "Documents\PrintedDocument.pdf"
' Specify that the first three pages should be printed.
printerSettings.PrintRange = PrintRange.SomePages
printerSettings.FromPage = 1
printerSettings.ToPage = 3
' Set the number of copies to print.
printerSettings.Copies = 1
' Print the workbook using the specified printer settings.
workbook.Print(printerSettings)
```

To print a specific sheet in a workbook, use the `DevExpress.Spreadsheet.Sheet.Print` method.

To define general page options, use properties of the `DevExpress.Spreadsheet.WorksheetView` object accessible from the `DevExpress.Spreadsheet.Worksheet.ActiveView` property. `WorksheetView` enables you to specify page orientation, margins and paper size settings.

The `DevExpress.Spreadsheet.WorksheetPrintOptions` object's properties allow you to specify more print-specific options, which include scaling, printing gridlines, titles, row and column headings, setting page order and more.

For more information on how to specify print settings and print a workbook, refer to the [Printing](#) example section.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.Print Overload List](#)

[DevExpress.Spreadsheet.WorksheetView](#)

[DevExpress.Spreadsheet.WorksheetPrintOptions](#)

[Printing](#)

Protect Method

Protects the structure and windows of a workbook.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Protect(  
    ByVal password As String,  
    ByVal lockStructure As Boolean,  
    ByVal lockWindows As Boolean  
)
```

C#

```
public void Protect(  
    string password,  
    bool lockStructure,  
    bool lockWindows  
)
```

Parameters

password
A string that specifies a password for the workbook. If an empty string is specified, the workbook can be unprotected without a password.

lockStructure
true, to lock the structure of the workbook (the position of the sheets); otherwise, **false**.

lockWindows
true, to prevent users from changing the position of worksheet windows; otherwise, **false**.

Remarks

Locked structure of the workbook prevents users from adding or deleting worksheets, from hiding worksheets or displaying hidden worksheets. Locked windows option prevents users from freezing or unfreezing panes.

If the workbook is already protected (the [IsProtected](#) property is **true**), the **Protect** method raises an exception.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4832>.

C#

```
(ProtectionActions.cs)  
// Protect workbook structure (prevents users from adding or deleting worksheets  
// or from displaying hidden worksheets).  
workbook.BeginUpdate();  
if (!workbook.IsProtected)  
    workbook.Protect("password", true, false);  
workbook.Worksheets[0].Visible = false;  
Worksheet worksheet = workbook.Worksheets[1];  
worksheet["D5"].Value = "You are not allowed to add or delete a worksheet.";  
worksheet["D6"].Value = "Hidden worksheets cannot be displayed.";  
workbook.EndUpdate();
```

Visual Basic

```
(ProtectionActions.vb)
' Protect workbook structure (prevents users from adding or deleting worksheets
' or from displaying hidden worksheets).
workbook.BeginUpdate()
If Not workbook.IsProtected Then
    workbook.Protect("password", True, False)
End If
workbook.Worksheets(0).Visible = False
Dim worksheet As Worksheet = workbook.Worksheets(1)
worksheet("D5").Value = "You are not allowed to add or delete a worksheet."
worksheet("D6").Value = "Hidden worksheets cannot be displayed."
workbook.EndUpdate()
```

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

RemoveService Method

Removes the service of the specified type from the service container.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
void RemoveService(Type serviceType)	Removes the service of the specified type from the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
void RemoveService(Type serviceType, bool promote)	Removes the service of the specified type from the service container. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also
[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.RemoveService Overload List](#)

Removes the service of the specified type from the service container.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub RemoveService(  
    ByVal serviceType As Type  
)
```

C#

```
public void RemoveService(  
    Type serviceType  
)
```

Parameters

serviceType
The type of service to remove.

Remarks

The Workbook implements the System.IServiceProvider and System.ComponentModel.Design.IServiceContainer interfaces. In addition to providing services, it also provides a mechanism for adding and removing services. To obtain a service, call the [GetService](#) method.

See Also
[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.RemoveService Overload List](#)
[GetService](#)

Removes the service of the specified type from the service container.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub RemoveService(  
    ByVal serviceType As Type,  
    ByVal promote As Boolean  
)
```

C#

```
public void RemoveService(  
    Type serviceType,  
    bool promote  
)
```

Parameters

serviceType

The type of service to remove.

promote

true, to promote this request to any parent service containers; **otherwise**, false.

Remarks

The Workbook implements the System.IServiceProvider and System.ComponentModel.Design.IServiceContainer interfaces. In addition to providing services, it also provides a mechanism for adding and removing services. To obtain a service, call the [GetService](#) method.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.RemoveService Overload List](#)

[GetService](#)

ReplaceService<T> Method

Performs a service substitution.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function ReplaceService(of T) (  
    ByVal newService As T  
) As T
```

C#

```
public T ReplaceService<T>(  
    T newService  
)
```

Parameters

newService

A service of the specified type that will be registered.

Return Value

Previously registered service of the specified type, or null (**Nothing** in Visual Basic) if the service does not exist.

Remarks

Use the **ReplaceService<T>** method instead of the [GetService](#) -> [RemoveService](#) -> [AddService](#) method sequence.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[GetService](#)

[RemoveService](#)

[AddService](#)

SaveDocument Method

Saves the document to the specified file. The file format is identified by the file extension.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
void SaveDocument(string fileName)	<p>Saves the document to the specified file. The file format is identified by the file extension.</p> <p>Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
Byte[] SaveDocument(DocumentFormat format)	<p>Saves a document to an array of bytes in the specified format.</p>
void SaveDocument(string fileName, DocumentFormat format)	<p>Saves the document to a file, specifying the document format.</p> <p>Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>
void SaveDocument(Stream stream, DocumentFormat format)	<p>Saves the workbook to a stream, specifying the export format.</p> <p>Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.</p>

See Also

[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.SaveDocument Overload List](#)

Saves the document to the specified file. The file format is identified by the file extension.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub SaveDocument(  
    ByVal fileName As String  
)
```

C#

```
public void SaveDocument(  
    string fileName  
)
```

Parameters

fileName
A string value specifying the path to a file into which to save the document.

Remarks

Use the **Workbook.Options.Export** property that returns the `DevExpress.XtraSpreadsheet.Export.WorkbookExportOptions` object to set global export options, or handle the [BeforeExport](#) event to specify options for an individual export action.

See Also[Workbook Class](#)[Workbook Members](#)[DevExpress.Spreadsheet Namespace](#)[Workbook.SaveDocument Overload List](#)[LoadDocument](#)[How to: Save a Document to a File](#)[How to: Export a Workbook to PDF](#)

Saves a document to an array of bytes in the specified format.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function SaveDocument(  
    ByVal format As DocumentFormat  
) As Byte[]
```

C#

```
public Byte[] SaveDocument(  
    DocumentFormat format  
)
```

Parameters

format

A `DevExpress.Spreadsheet.DocumentFormat` enumeration member specifying the format of the document to be saved.

Return Value

A `System.Byte[]` object that is an array of bytes containing document data in the specified format.

Remarks

The **SaveDocument** method can be used to store a workbook in an external database. Refer to the [How to: Store a Workbook in the Database](#) document for additional information.

See Also[Workbook Class](#)[Workbook Members](#)[DevExpress.Spreadsheet Namespace](#)[Workbook.SaveDocument Overload List](#)[LoadDocument](#)

Saves the document to a file, specifying the document format.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub SaveDocument(  
    ByVal fileName As String,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void SaveDocument(  
    string fileName,  
    DocumentFormat format  
)
```

Parameters

fileName

A string value specifying the path to a file into which to save the document.

format

One of the `DevExpress.Spreadsheet.DocumentFormat` enumeration values.

Remarks

Use the **Workbook.Options.Export** property that returns the `DevExpress.XtraSpreadsheet.Export.WorkbookExportOptions` object to set global export options, or handle the [BeforeExport](#) event to specify options for an individual export action.

Example

Call the [SaveDocument](#) method with the passed file path to save a workbook to the file. Specify the file format as the second parameter of the method using the `DevExpress.Spreadsheet.DocumentFormat` enumerator.

```
C#

// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
// ...
Workbook workbook = new Workbook();
// ...
// Save the modified document to the file.
workbook.SaveDocument("Documents\\SavedDocument.xlsx", DocumentFormat.Xlsx);
```

```
Visual Basic

' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
' ...
Private workbook As New Workbook()
' ...
' Save the modified document to the file.
workbook.SaveDocument("Documents\\SavedDocument.xlsx", DocumentFormat.Xlsx)
```

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.SaveDocument Overload List](#)

[LoadDocument](#)

[How to: Save a Document to a File](#)

[How to: Export a Workbook to PDF](#)

Saves the workbook to a stream, specifying the export format.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: `DevExpress.Docs.v18.1.dll`

Syntax

Visual Basic

```
Public Sub SaveDocument(  
    ByVal stream As Stream,  
    ByVal format As DocumentFormat  
)
```

C#

```
public void SaveDocument(  
    Stream stream,  
    DocumentFormat format  
)
```

Parameters

stream

The [System.IO.Stream](#) object to output the document to.

format

One of the `DevExpress.Spreadsheet.DocumentFormat` values that is the format of the exported document.

Remarks

Take into account the following when saving a document to a stream:

- Do not save a workbook to a stream that does not support seek operations. An exception is thrown in this case. Use the **`System.IO.Stream.CanSeek`** property to determine if the stream supports seeking.
- Do not save a workbook to the same stream from which it was loaded. In this case, document bytes are written to the end of the existing stream and an invalid document is produced.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Workbook.SaveDocument Overload List](#)

[How to: Save a Document to a File](#)

Search Method

Performs a search in the current document using the default parameters.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
IEnumerable<Cell> Search(string text)	Performs a search in the current document using the default parameters. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
IEnumerable<Cell> Search(string text, SearchOptions options)	Performs a search in the current document using specified options. Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also
[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.Search Overload List](#)

Performs a search in the current document using the default parameters.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Search(  
    ByVal text As String  
) As IEnumerable(of Cell)
```

C#

```
public IEnumerable<Cell> Search(  
    string text  
)
```

Parameters

text
A [System.String](#) representing the search text.

Return Value

An object implementing the [System.Collections.IEnumerable](#) interface which is the collection of cells that match the search term.

Remarks

Use the **Search** method to perform a search in the current document with the default parameters listed in the table below.

Option	Default Value
Look In	Values and Formulas

Search Direction	By Rows
Match Case	False
Match Entire Cell Contents	False

To restrict a search to the active worksheet or a cell range, use the `DevExpress.Spreadsheet.Worksheet.Search` or `DevExpress.Spreadsheet.Range.Search` methods. For more information about the search functionality in the `SpreadsheetControl`, refer to the [Find and Replace](#) article.

See Also
[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.Search Overload List](#)
[Find and Replace](#)

Performs a search in the current document using specified options.

Use of this property in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: `DevExpress.Docs.v18.1.dll`

Syntax

Visual Basic

```
Public Function Search(  
    ByVal text As String,  
    ByVal options As SearchOptions  
) As IEnumerable(of Cell)
```

C#

```
public IEnumerable<Cell> Search(  
    string text,  
    SearchOptions options  
)
```

Parameters

text
A [System.String](#) representing the search text.

options
A `DevExpress.Spreadsheet.SearchOptions` instance containing required search options.

Return Value

An object implementing the [System.Collections.IEnumerable](#) interface which is the collection of cells that match the search term.

Remarks

To restrict a search to the active worksheet or a cell range, use the `DevExpress.Spreadsheet.Worksheet.Search` or `DevExpress.Spreadsheet.Range.Search` methods. For more information about the search functionality in the `SpreadsheetControl`, refer to the [Find and Replace](#) article.

See Also
[Workbook Class](#)
[Workbook Members](#)
[DevExpress.Spreadsheet Namespace](#)
[Workbook.Search Overload List](#)
[Find and Replace](#)

Unprotect Method

Removes protection from a workbook.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Unprotect(  
    ByVal password As String  
) As Boolean
```

C#

```
public bool Unprotect(  
    string password  
)
```

Parameters

password

A string that specifies the password with which the workbook is protected.

Return Value

true, if protection is successfully removed; otherwise, **false**.

Remarks

If the workbook is not protected, the **Unprotect** method raises an exception. Use the [IsProtected](#) property to check whether the workbook is already protected.

See Also

[Workbook Class](#)

[Workbook Members](#)

[DevExpress.Spreadsheet Namespace](#)

[Protect](#)

[IsProtected](#)

WorkbookExtensions Class

Defines extension methods for objects exposing the `DevExpress.Spreadsheet.IWorkbook` interface.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

`NotInheritable Public Class WorkbookExtensions Inherits Object`

C#

`public sealed abstract class WorkbookExtensions : object`

Remarks

To enable workbook extensions, add a reference to the **DevExpress.Docs.v18.1.dll** assembly and explicitly import the [DevExpress.Spreadsheet](#) namespace into the code with a **using** directive (**Imports** in Visual Basic).

Subsequently, all extension methods become accessible as methods of objects, exposing the `DevExpress.Spreadsheet.IWorkbook` interface (`DevExpress.XtraSpreadsheet.SpreadsheetControl.Document` or non-visual [Workbook](#)) and you can call them using instance method syntax.

Use of these methods in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Refer to the [DevExpress Subscription](#) page for pricing information.

Inheritance Hierarchy

[Object](#)

WorkbookExtensions

See Also

[WorkbookExtensions Members](#)









[DevExpress.Spreadsheet Namespace](#)

WorkbookExtensions Members

Defines extension methods for objects exposing the `DevExpress.Spreadsheet.IWorkbook` interface.

The following tables list the members exposed by the [WorkbookExtensions](#) type.

Public Methods

	Name	Description
	Clone	Overloaded. Creates a workbook copy. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Merge	Merges data from multiple workbooks into a single workbook. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[WorkbookExtensions Members](#)








[DevExpress.Spreadsheet Namespace](#)

WorkbookExtensions Methods

Defines extension methods for objects exposing the `DevExpress.Spreadsheet.IWorkbook` interface.

The following tables list the members exposed by the [WorkbookExtensions](#) type.

Public Methods

	Name	Description
	Clone	Overloaded. Creates a workbook copy. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Merge	Merges data from multiple workbooks into a single workbook. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[WorkbookExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

Clone Method

Creates a workbook copy.
Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
static Workbook Clone(IWorkbook workbook)	Creates a workbook copy. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static Workbook Clone(IWorkbook workbook, bool copyFormulas)	Creates a workbook copy and specifies whether the resulting document should contain formula results instead of formula expressions. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also
[WorkbookExtensions Class](#)
[WorkbookExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorkbookExtensions.Clone Overload List](#)

Creates a workbook copy.
Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Clone(  
    ByVal workbook As IWorkbook  
) As Workbook
```

C#

```
public static Workbook Clone(  
    IWorkbook workbook  
)
```

Parameters

workbook
An object exposing the DevExpress.Spreadsheet.IWorkbook interface that specifies the source workbook for copying.

Return Value

A [Workbook](#) object that is the created copy.

Remarks

The **Clone** method is an extension method of the object that exposes the DevExpress.Spreadsheet.IWorkbook interface (DevExpress.XtraSpreadsheet.SpreadsheetControl.Document or non-visual [Workbook](#)) and is called by using instance method syntax.

The following example demonstrates how to create copies of the existing workbooks.

C#

```
// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
// ...
// Create a new Workbook object.
Workbook workbook = new Workbook();
workbook.LoadDocument("Document.xlsx", DocumentFormat.Xlsx);
// Create a copy of the Workbook object.
Workbook copy = workbook.Clone();
// Create a copy of the document loaded into the SpreadsheetControl.
Workbook copy2 = spreadsheetControl.Document.Clone();
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
' ...
' Create a new Workbook object.
Private workbook As New Workbook()
workbook.LoadDocument("Document.xlsx", DocumentFormat.Xlsx)
' Create a copy of the Workbook object.
Dim copy As Workbook = workbook.Clone()
' Create a copy of the document loaded into the SpreadsheetControl.
Dim copy2 As Workbook = spreadsheetControl.Document.Clone()
```

See Also

[WorkbookExtensions Class](#)
[WorkbookExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorkbookExtensions.Clone Overload List](#)

Creates a workbook copy and specifies whether the resulting document should contain formula results instead of formula expressions.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Clone(
    ByVal workbook As IWorkbook,
    ByVal copyFormulas As Boolean
) As Workbook
```

C#

```
public static Workbook Clone(
    IWorkbook workbook,
    bool copyFormulas
)
```

Parameters

workbook

An object exposing the DevExpress.Spreadsheet.IWorkbook interface that specifies the source workbook for copying.

copyFormulas

true, to copy formulas of the source workbook; otherwise, **false**. If **false**, only the calculated results are copied.

Return Value

A [Workbook](#) object that is the created copy.

Remarks

The **Clone** method is an extension method of the object that exposes the DevExpress.Spreadsheet.IWorkbook interface (DevExpress.XtraSpreadsheet.SpreadsheetControl.Document or non-visual [Workbook](#)) and is called by using instance method syntax.

Use the current **Clone** method overload to create a copy of the specified workbook without formulas: the resulting document retains only the formula results.

See Also[WorkbookExtensions Class](#)[WorkbookExtensions Members](#)[DevExpress.Spreadsheet Namespace](#)[WorkbookExtensions.Clone Overload List](#)

Merge Method

Merges data from multiple workbooks into a single workbook.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function Merge(  
    ByVal workbook As IWorkbook,  
    ByVal options As WorkbookMergeOptions,  
    ByVal workbooks As IWorkbook[]  
) As IWorkbookMergeResult
```

C#

```
public static IWorkbookMergeResult Merge(  
    IWorkbook workbook,  
    WorkbookMergeOptions options,  
    params IWorkbook[] workbooks  
)
```

Parameters

- workbook*
An object exposing the DevExpress.Spreadsheet.IWorkbook interface that specifies a workbook to be merged.
- options*
A DevExpress.Spreadsheet.WorkbookMergeOptions object containing merge options.
- workbooks*
An array of DevExpress.Spreadsheet.IWorkbook objects that are workbooks that should be merged with a master workbook.

Return Value

An object exposing the DevExpress.Spreadsheet.IWorkbookMergeResult interface that contains the result of a merge process.

Remarks

The **Merge** method is an extension method of the object that exposes the DevExpress.Spreadsheet.IWorkbook interface (DevExpress.XtraSpreadsheet.SpreadsheetControl.Document or non-visual [Workbook](#)) and is called by using instance method syntax.

Use the **Merge** method to merge multiple workbooks into a single document. The DevExpress.Spreadsheet.WorkbookMergeOptions object allows you to specify merge options. Set the DevExpress.Spreadsheet.WorkbookMergeOptions.CreateNewWorkbook property to **true** to combine all workbooks into a new document. By default, this property is **false**, so a new document is not created and all worksheets from the specified *workbooks* are added to a master workbook that calls this method.

If a workbook being merged contains a worksheet with the same name as a worksheet that already exists in the master workbook, the added worksheet is automatically renamed and all cell references containing this worksheet name are updated to use a new name. For instance, if you're trying to merge workbooks containing worksheets named "Sheet1" with a master workbook where "Sheet1" already exists, these new worksheets will be renamed in the following way: "Sheet1 (2)", "Sheet1 (3)", etc. To rename a worksheet, use the DevExpress.Spreadsheet.Worksheet.Name property of the corresponding DevExpress.Spreadsheet.Worksheet object.

The following example demonstrates how to merge a document loaded into the **SpreadsheetControl** with a [Workbook](#) instance.

C#

```
// Add a reference to the DevExpress.Docs.dll assembly.
using DevExpress.Spreadsheet;
// ...
// Create a new Workbook object.
Workbook book1 = new Workbook();
book1.LoadDocument("Document1.xlsx", DocumentFormat.Xlsx);
IWorkbook book2 = spreadsheetControl.Document;
// Load a document into SpreadsheetControl.
book2.LoadDocument("Document2.xlsx", DocumentFormat.Xlsx);
// Combine the loaded document with "Document1".
book2.Merge(WorkbookMergeOptions.Default, book1);
```

Visual Basic

```
' Add a reference to the DevExpress.Docs.dll assembly.
Imports DevExpress.Spreadsheet
' ...
' Create a new Workbook object.
Private book1 As New Workbook()
book1.LoadDocument("Document1.xlsx", DocumentFormat.Xlsx)
Dim book2 As IWorkbook = spreadsheetControl.Document
' Load a document into SpreadsheetControl.
book2.LoadDocument("Document2.xlsx", DocumentFormat.Xlsx)
' Combine the loaded document with "Document1".
book2.Merge(WorkbookMergeOptions.Default, book1)
```

See Also

[WorkbookExtensions Class](#)

[WorkbookExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

WorksheetExtensions Class

Defines extension methods for the `DevExpress.Spreadsheet.Worksheet` class.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

`NotInheritable Public Class WorksheetExtensions Inherits Object`

C#

`public sealed abstract class WorksheetExtensions : object`

Remarks

To enable worksheet extensions, add the reference to the DevExpress.Docs.v18.1.dll assembly and explicitly import the [DevExpress.Spreadsheet](#) namespace into the code with a **using** directive (**Imports** in Visual Basic).

Subsequently, all extension methods become accessible and you can call them using the common instance method syntax.

Use of these methods in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Refer to the [DevExpress Subscription](#) page for pricing information.

Inheritance Hierarchy

[Object](#)

WorksheetExtensions

See Also

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

WorksheetExtensions Members

Defines extension methods for the `DevExpress.Spreadsheet.Worksheet` class.

The following tables list the members exposed by the [WorksheetExtensions](#) type.

Public Methods

	Name	Description
	CreateDataTable	Overloaded. Creates an empty data table from the specified cell range. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CreateDataTableExporter	Creates an instance of the data table exporter. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Import	Overloaded. Imports data from an one-dimensional array of boolean values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

WorksheetExtensions Methods

Defines extension methods for the `DevExpress.Spreadsheet.Worksheet` class.

The following tables list the members exposed by the [WorksheetExtensions](#) type.

Public Methods

	Name	Description
	CreateDataTable	Overloaded. Creates an empty data table from the specified cell range. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	CreateDataTableExporter	Creates an instance of the data table exporter. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	Import	Overloaded. Imports data from an one-dimensional array of boolean values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

CreateDataTable Method

Creates an empty data table from the specified cell range.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
static DataTable CreateDataTable(Worksheet sheet, Range range, bool rangeHasHeaders)	Creates an empty data table from the specified cell range. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static DataTable CreateDataTable(Worksheet sheet, Range range, bool rangeHasHeaders, bool stringColumnType)	Creates an empty data table from the specified cell range. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also
[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.CreateDataTable Overload List](#)

Creates an empty data table from the specified cell range.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function CreateDataTable(  
    ByVal sheet As Worksheet,  
    ByVal range As Range,  
    ByVal rangeHasHeaders As Boolean  
) As DataTable
```

C#

```
public static DataTable CreateDataTable(  
    Worksheet sheet,  
    Range range,  
    bool rangeHasHeaders  
)
```

Parameters

sheet
A DevExpress.Spreadsheet.Worksheet object that is a worksheet containing the specified range.

range
A DevExpress.Spreadsheet.Range of cells that will be transformed to a data table.

rangeHasHeaders
true, to use the content of the first row in a range as column names in a data table; otherwise, **false**.

Return Value

A [System.Data.DataTable](#) object that is the resulting data table.

Remarks

The **CreateDataTable** method creates an empty data table. The number of columns in a data table is the same as the number of columns in the specified range. The first row of the specified cell range defines column names if the **rangeHasHeaders** parameter is set to **true**; otherwise the first row is used to determine the column data types.

If the first row in a specified range contains column names (the **rangeHasHeaders** parameter for the **CreateDataTable** method is set to **true**), the second row is used to determine the column data types.

To determine the column data type for the resulting data table, the `DevExpress.Spreadsheet.CellValueType` of data contained in a cell is used.

Note

During export, if the cell value type does not match the type of the column in the resulting data table, the exception **Conversion error (Subscribe to the `CellValueConversionError` event)** may occur. To prevent this, you can validate cell value types and handle the `DevExpress.Spreadsheet.Export.DataTableExporter.CellValueConversionError` event, as illustrated in the following example. Another workaround is to use the **WorksheetExtensions.CreateDataTable(Worksheet, Range, Boolean, Boolean)** method, which treats all cell values as text. In this case, all columns in the resulting table will have the string type.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4997>.

	A	B	C	D	E	F	G	H	I
4		This report shows trade data on the United States on a country by country basis. All values are in million USD.							
5		Country	Exports	Imports	Balance	Exports 1Y	Imports 1Y	Balance 1Y	As Of
6		Australia	2,277.90	815.40	▲ 1,462.50	-8.94%	-2.79%	-0.12%	Jul-13
7		Belgium	2,808.90	1,826.20	▲ 982.70	19.23%	22.91%	0.13%	May-13
8		Brazil	3,423.20	2,571.40	▲ 851.80	-4.64%	18.83%	-0.40%	May-13
9		Canada	26,453.10	28,343.10	▼ -1,890.00	18.87%	9.94%	-0.46%	May-13
10		China	8,786.50						
11		France	2,642.30						
12		Germany	4,052.50						
13		Hong Kong	3,385.40						
14		India	1,935.80						
		Country	Exports	Imports	Balance	Exports 1Y	Imports 1Y	Balance 1Y	As Of
		Australia	2277.9	815.4	1462.5	-0.0894	-0.0279	-0.0012	7/1/2013
		Belgium	2808.9	1826.2	982.7	0.1923	0.2291	0.0013	5/1/2013
		Brazil	3423.2	2571.4	851.8	-0.0464	0.1883	-0.004	5/1/2013
		Canada	26453.1	28343.1	-1890	0.1887	0.0994	-0.0046	5/1/2013
		China	8786.5	36646.2	-27859.7	-0.1478	0.0518	0.0014	5/1/2013
		France	2642.3	3563.7	-921.4	0.0882	0.063	0	5/1/2013
		Germany	4052.5	9888.3	-5835.8	0.0662	0.052	0.0004	5/1/2013
		Hong Kong	3385.4	413	2972.4	-0.2327	0.0034	-0.0026	
		India	1935.8	4201.5	-2265.7				

C#

```

(Form1.cs)
using DevExpress.Spreadsheet;
using DevExpress.Spreadsheet.Export;

Worksheet worksheet = spreadsheetControl1.Document.Worksheets.ActiveWorksheet;
Range range = worksheet.Selection;
bool rangeHasHeaders = this.barCheckItemHasHeaders1.Checked;
// Create a data table with column names obtained from the first row in a range if it has headers
// Column data types are obtained from cell value types of cells in the first data row of the
DataTable dataTable = worksheet.CreateDataTable(range, rangeHasHeaders);
//Validate cell value types. If cell value types in a column are different, the column values
for (int col = 0; col < range.ColumnCount; col++)
{
    CellValueType cellType = range[0, col].Value.Type;
    for (int r = 1; r < range.RowCount; r++)
    {
        if (cellType != range[r, col].Value.Type)
        {
            dataTable.Columns[col].DataType = typeof(string);
            break;
        }
    }
}
// Create the exporter that obtains data from the specified range,
// skips the header row (if required) and populates the previously created data table.
DataTableExporter exporter = worksheet.CreateDataTableExporter(range, dataTable, rangeHasHeaders);
// Handle value conversion errors.
exporter.CellValueConversionError += exporter_CellValueConversionError;
// Perform the export.
exporter.Export();
void exporter_CellValueConversionError(object sender, CellValueConversionEventArgs e)
{
    MessageBox.Show("Error in cell " + e.Cell.GetReferenceA1());
    e.DataTableValue = null;
    e.Action = DataTableExporterAction.Continue;
}

```

Visual Basic

```

(Form1.vb)
Imports DevExpress.Spreadsheet
Imports DevExpress.Spreadsheet.Export
    Dim worksheet As Worksheet = spreadsheetControll1.Document.Worksheets.ActiveWorksheet
    Dim range As Range = worksheet.Selection
    Dim rangeHasHeaders As Boolean = Me.barCheckItemHasHeaders1.Checked
    ' Create a data table with column names obtained from the first row in a range if it has headers
    ' Column data types are obtained from cell value types of cells in the first data row of the range
    Dim dataTable As DataTable = worksheet.CreateDataTable(range, rangeHasHeaders)
    ' Validate cell value types. If cell value types in a column are different, the column values are converted to strings
    For col As Integer = 0 To range.ColumnCount - 1
        Dim cellType As CellValueType = range(0, col).Value.Type
        For r As Integer = 1 To range.RowCount - 1
            If cellType <> range(r, col).Value.Type Then
                dataTable.Columns(col).DataType = GetType(String)
                Exit For
            End If
        Next r
    Next col
    ' Create the exporter that obtains data from the specified range,
    ' skips the header row (if required) and populates the previously created data table.
    Dim exporter As DataTableExporter = worksheet.CreateDataTableExporter(range, dataTable, rangeHasHeaders)
    ' Handle value conversion errors.
    AddHandler exporter.CellValueConversionError, AddressOf exporter_CellValueConversionError
    ' Perform the export.
    exporter.Export()
    Private Sub exporter_CellValueConversionError(ByVal sender As Object, ByVal e As CellValueConversionErrorEventArgs)
        MessageBox.Show("Error in cell " & e.Cell.GetReferenceA1())
        e.DataTableValue = Nothing
        e.Action = DataTableExporterAction.Continue
    End Sub

```

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.CreateDataTable Overload List](#)

Creates an empty data table from the specified cell range.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

static Public Function CreateDataTable(
    ByVal sheet As Worksheet,
    ByVal range As Range,
    ByVal rangeHasHeaders As Boolean,
    ByVal stringColumnType As Boolean
) As DataTable

```

C#

```

public static DataTable CreateDataTable(
    Worksheet sheet,
    Range range,
    bool rangeHasHeaders,
    bool stringColumnType
)

```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet object that is a worksheet containing the specified range.

range

A `DevExpress.Spreadsheet.Range` of cells that will be transformed to a data table.

rangeHasHeaders

true, to use the content of the first row in a range as column names in a data table; otherwise, **false**.

stringColumnType

true, to use **string** data type for all columns in the resulting data table; otherwise, **false**.

Return Value

A [System.Data.DataTable](#) object that is the resulting data table.

Remarks

The **CreateDataTable** method creates an empty data table. The number of columns in a data table is the same as the number of columns in the specified range. The first row of the specified cell range defines column names if the **rangeHasHeaders** parameter is set to **true**. If the **stringColumnType** parameter is **true**, all columns in a resulting data table will be of the **string** type.

The first row in the specified range is used to determine column data types if the **rangeHasHeaders** and the **stringColumnType** parameters are set to **false**.

If the first row in a specified range contains column names (the **rangeHasHeaders** parameter for the **CreateDataTable** method is set to **true**), the second row is used to determine the column data types.

To determine the column data type for the resulting data table, the `DevExpress.Spreadsheet.CellValue.Type` of data contained in a cell is used.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.CreateDataTable Overload List](#)

CreateDataTableExporter Method

Creates an instance of the data table exporter.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Function CreateDataTableExporter(  
    ByVal sheet As Worksheet,  
    ByVal range As Range,  
    ByVal dataTable As DataTable,  
    ByVal rangeHasHeaders As Boolean  
) As DataTableExporter
```

C#

```
public static DataTableExporter CreateDataTableExporter(  
    Worksheet sheet,  
    Range range,  
    DataTable dataTable,  
    bool rangeHasHeaders  
)
```

Parameters

- sheet*
A DevExpress.Spreadsheet.Worksheet interface for the worksheet which contains data for export.
- range*
A DevExpress.Spreadsheet.Range of cells containing data for export.
- dataTable*
A [System.Data.DataTable](#) object which will be populated with exported data.
- rangeHasHeaders*
true, to skip the first row in the specified range; otherwise, **false**.

Return Value

A DevExpress.Spreadsheet.Export.DataTableExporter instance that performs export to a data table.

Remarks

Use the **CreateDataTableExporter** method to create a DevExpress.Spreadsheet.Export.DataTableExporter which is the key object of the data export.

Note

When calling the **CreateDataTableExporter** method, make sure that the number of data columns in the specified *dataTable* is the same as the number of columns in the specified *range*; otherwise, a **System.ArgumentException** will be thrown.

Therefore, if you need to add an additional column to the data table (e.g., an auto-incremented or computed column), you should do it only after DevExpress.Spreadsheet.Export.DataTableExporter is created.

Use the [CreateDataTable](#) method to create a data table that corresponds to the specified worksheet cell range.

See Also

- [WorksheetExtensions Class](#)
- [WorksheetExtensions Members](#)
- [DevExpress.Spreadsheet Namespace](#)

Import Method

Imports data from a two-dimensional array of strings.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Overload List

Name	Description
static void Import(Worksheet sheet, String[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)	Imports data from a two-dimensional array of strings. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Int16[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)	Imports data from a two-dimensional array of integers. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, object dataSource, int firstRowIndex, int firstColumnIndex)	Imports data from a data source. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Int32[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)	Imports data from a two-dimensional array of integers. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Byte[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)	Imports data from a two-dimensional array of bytes. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Int64[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)	Imports data from a two-dimensional array of long integers. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Object[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)	Imports data from a two-dimensional array of objects. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Single[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)	Imports data from a two-dimensional array of single-precision floating point values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Double[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)	Imports data from a two-dimensional array of double-precision floating point values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

<u>static void Import(Worksheet sheet, Decimal[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)</u>	Imports data from a two-dimensional array of decimal values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, DateTime[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)</u>	Imports data from a two-dimensional array of DateTime values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, Boolean[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex)</u>	Imports data from a two-dimensional array of boolean values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, Byte[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)</u>	Imports data from an one-dimensional array of bytes. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, DataTable source, bool addHeader, int firstRowIndex, int firstColumnIndex)</u>	Imports data from a data table. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, Boolean[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)</u>	Imports data from an one-dimensional array of boolean values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, IDataReader source, bool addHeader, int firstRowIndex, int firstColumnIndex)</u>	Imports data from a relational database. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, IEnumerable source, int firstRowIndex, int firstColumnIndex, bool isVertical)</u>	Imports data from a collection. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, Object[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)</u>	Imports data from an one-dimensional array of objects. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, String[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)</u>	Imports data from an one-dimensional array of strings. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
<u>static void Import(Worksheet sheet, Object[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex, IDataValueConverter converter)</u>	Imports data from a two-dimensional array of objects.

	Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Int64[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)	Imports data from an one-dimensional array of long integers. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Decimal[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)	Imports data from an one-dimensional array of decimal values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Single[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)	Imports data from an one-dimensional array of single-precision floating point values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, DateTime[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)	Imports data from an one-dimensional array of DateTime values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, String[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex, DataImportOptions options)	Imports data from a two-dimensional array of strings. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Object[,] twoDimensionalArray, int firstRowIndex, int firstColumnIndex, DataImportOptions options)	Imports data from a two-dimensional array of objects. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, object dataSource, int firstRowIndex, int firstColumnIndex, DataSourceImportOptions options)	Imports data from a data source. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Int16[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)	Imports data from an one-dimensional array of short integers. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Double[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)	Imports data from an one-dimensional array of double-precision floating point values. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Int32[] array, int firstRowIndex, int firstColumnIndex, bool isVertical)	Imports data from an one-dimensional array of integers.

	Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, IDataReader source, bool addHeader, int firstRowIndex, int firstColumnIndex, IDataValueConverter converter)	Imports data from a relational database. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Object[] array, int firstRowIndex, int firstColumnIndex, bool isVertical, IDataValueConverter converter)	Imports data from an one-dimensional array of objects. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, Object[] array, int firstRowIndex, int firstColumnIndex, bool isVertical, DataImportOptions options)	Imports data from an one-dimensional array of objects. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, DataTable source, bool addHeader, int firstRowIndex, int firstColumnIndex, IDataValueConverter converter)	Imports data from a data table. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, DataTable source, bool addHeader, int firstRowIndex, int firstColumnIndex, DataImportOptions options)	Imports data from a data table. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, IEnumerable source, int firstRowIndex, int firstColumnIndex, bool isVertical, DataImportOptions options)	Imports data from a collection. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, IDataReader source, bool addHeader, int firstRowIndex, int firstColumnIndex, DataImportOptions options)	Imports data from a relational database. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
static void Import(Worksheet sheet, IEnumerable source, int firstRowIndex, int firstColumnIndex, bool isVertical, IDataValueConverter converter)	Imports data from a collection. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

See Also

[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)

Imports data from a two-dimensional array of strings.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As String[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    String[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of strings that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of integers.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Int16[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Int16[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of integers that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from a data source.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal dataSource As Object,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    object dataSource,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

dataSource

An object that is the data source for import.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

Remarks

The **Import** method attempts to use the specified object as the data source.

See Also

[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from a two-dimensional array of integers.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Int32[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Int32[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of integers that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of bytes.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Byte[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Byte[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of bytes that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of long integers.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Int64[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Int64[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of long integers that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of objects.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Object[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Object[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet
A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray
A two-dimensional array of objects that is the source of data.

firstRowIndex
An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex
An integer that is the column index of the start cell in which the imported data will be inserted.

Remarks

The following code snippet illustrates how to use the **Import** method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4751>.

C#

```
(Form1.cs)
// Create an array containing string values.
string[] array = new string[] { "AAA", "BBB", "CCC", "DDD" };
// Import the array into the worksheet and insert it horizontally, starting with the B1 cell.
worksheet.Import(array, 0, 1, false);
// Create a two-dimensional array containing string values.
String[,] names = new String[2, 4]{
    {"Ann", "Edward", "Angela", "Alex"},
    {"Rachel", "Bruce", "Barbara", "George"}
};
// Import the two-dimensional array into the worksheet and insert it, starting with the B3 cell.
worksheet.Import(names, 2, 1);
```

Visual Basic

```
(Form1.vb)
' Create an array containing string values.
Dim array() As String = { "AAA", "BBB", "CCC", "DDD" }
' Import the array into the worksheet and insert it horizontally, starting with the B1 cell.
worksheet.Import(array, 0, 1, False)
' Create a two-dimensional array containing string values.
Dim names(,) As String = {
    {"Ann", "Edward", "Angela", "Alex"},
    {"Rachel", "Bruce", "Barbara", "George"}
}
' Import the two-dimensional array into the worksheet and insert it, starting with the B3 cell.
worksheet.Import(names, 2, 1)
```

The image below shows the results.

	A	B	C	D	E
1	Import an array horizontally:	AAA	BBB	CCC	DDD
2					
3	Import a two-dimensional array:	Ann	Edward	Angela	Alex
4		Rachel	Bruce	Barbara	George
5					
6					

See Also
[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from a two-dimensional array of single-precision floating point values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Single[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Single[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of single-precision floating point values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of double-precision floating point values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Double[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Double[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of double-precision floating point values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of decimal values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Decimal[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Decimal[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of decimal values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of DateTime values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As DateTime[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    DateTime[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of [System.DateTime](#) values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of boolean values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Boolean[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Boolean[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of boolean values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

See Also

[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from an one-dimensional array of bytes.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal array As Byte[],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Byte[] array,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of bytes that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from a data table.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal source As DataTable,  
    ByVal addHeader As Boolean,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer  
)
```

```
C#  
public static void Import(  
    Worksheet sheet,  
    DataTable source,  
    bool addHeader,  
    int firstRowIndex,  
    int firstColumnIndex  
)
```

Parameters

sheet
A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

source
A [System.Data.DataTable](#) object that is the data source for import.

addHeader
true, to insert column names to the row above the cells containing imported data; otherwise, **false**.

firstRowIndex
An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex
An integer that is the column index of the start cell in which the imported data will be inserted.

Remarks

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4751>.

The following code illustrates how to import data from a [System.Data.DataTable](#) object to a worksheet. Note that the cell data types are set automatically, according to the data types of the source column. Cell formats are set automatically to the default value for the cell data type. However, you can easily change them, as described in the [How to: Specify Number or Date Format for Cell Content](#) topic.

```
C#  
(Form1.cs)  
void ImportDataTable() {  
    Worksheet worksheet = spreadsheetControl1.Document.Worksheets[0];  
    // Create a "Products" DataTable object with four columns.  
    DataTable sourceTable = new DataTable("Products");  
    sourceTable.Columns.Add("Product", typeof(string));  
    sourceTable.Columns.Add("Price", typeof(float));  
    sourceTable.Columns.Add("Quantity", typeof(Int32));  
    sourceTable.Columns.Add("Discount", typeof(float));  
    sourceTable.Rows.Add("Chocolate", 5, 15, 0.03);  
    sourceTable.Rows.Add("Konbu", 9, 55, 0.1);  
    sourceTable.Rows.Add("Geitost", 15, 70, 0.07);  
    // Import data from the data table into the worksheet and insert it, starting with the B2 cell.  
    worksheet.Import(sourceTable, true, 1, 1);  
}
```

Visual Basic

```
(Form1.vb)
Private Sub ImportDataTable()
    Dim worksheet As Worksheet = spreadsheetControll.Document.Worksheets(0)
    ' Create a "Products" DataTable object with four columns.
    Dim sourceTable As New DataTable("Products")
    sourceTable.Columns.Add("Product", GetType(String))
    sourceTable.Columns.Add("Price", GetType(Single))
    sourceTable.Columns.Add("Quantity", GetType(Int32))
    sourceTable.Columns.Add("Discount", GetType(Single))
    sourceTable.Rows.Add("Chocolade", 5, 15, 0.03)
    sourceTable.Rows.Add("Konbu", 9, 55, 0.1)
    sourceTable.Rows.Add("Geitost", 15, 70, 0.07)
    ' Import data from the data table into the worksheet and insert it, starting with the B2 cell.
    worksheet.Import(sourceTable, True, 1, 1)
End Sub
```

The image below shows the results.

	A	B	C	D	E	F	G
1							
2		Product	Price	Quantity	Discount		
3		Chocolade	5	15	0.03		
4		Konbu	9	55	0.1		
5		Geitost	15	70	0.07		
6							
7							
8							

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of boolean values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(
    ByVal sheet As Worksheet,
    ByVal array As Boolean[],
    ByVal firstRowIndex As Integer,
    ByVal firstColumnIndex As Integer,
    ByVal isVertical As Boolean
)
```

C#

```
public static void Import(
    Worksheet sheet,
    Boolean[] array,
    int firstRowIndex,
    int firstColumnIndex,
    bool isVertical
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of boolean values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a relational database.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(
    ByVal sheet As Worksheet,
    ByVal source As IDataReader,
    ByVal addHeader As Boolean,
    ByVal firstRowIndex As Integer,
    ByVal firstColumnIndex As Integer
)
```

C#

```
public static void Import(
    Worksheet sheet,
    IDataReader source,
    bool addHeader,
    int firstRowIndex,
    int firstColumnIndex
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

source

An object that exposes the System.Data.IDataReader interface, e.g., an object returned by the SqlCommand.ExecuteReader method.

addHeader

true, to insert column names to the row above the cells containing imported data; otherwise, **false**.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex
An integer that is the column index of the start cell in which the imported data will be inserted.

Remarks

Use the **Import** method overload to import data from .NET Framework data providers that access relational databases, such as MS SQL Server.

See Also
[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from a collection.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal source As IEnumerable,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    IEnumerable source,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical  
)
```

Parameters

sheet
A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.
source
An object that exposes the [System.Collections.IEnumerable](#) interface provided by a collection of objects being imported.
firstRowIndex
An integer that is the row index of the start cell in which the imported data will be inserted.
firstColumnIndex
An integer that is the column index of the start cell in which the imported data will be inserted.
isVertical
true, to insert imported data vertically; otherwise, **false**

Remarks

The following code snippet illustrates how to use the **Import** method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4751>.

C#

```
(Form1.cs)
// Create a List object containing string values.
List<string> cities = new List<string>();
cities.Add("New York");
cities.Add("Rome");
cities.Add("Beijing");
cities.Add("Delhi");
// Import the list into the worksheet and insert it vertically, starting with the B1 cell.
worksheet.Import(cities, 0, 1, true);
```

```
Visual Basic
(Form1.vb)
' Create a List object containing string values.
Dim cities As New List(Of String) ()
cities.Add("New York")
cities.Add("Rome")
cities.Add("Beijing")
cities.Add("Delhi")
' Import the list into the worksheet and insert it vertically, starting with the B1 cell.
worksheet.Import(cities, 0, 1, True)
```

The image below shows the results.

	A	B	C	D	E
1	Import data from List vertically:	New York			
2		Rome			
3		Beijing			
4		Delhi			
5					
6					

See Also
[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from an one-dimensional array of objects.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal array As Object[],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Object[] array,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical  
)
```

Parameters

sheet
A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array
A one-dimensional array of objects that is the data source.

firstRowIndex
An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex
An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical
true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

The following code snippet illustrates how to use the **Import** method.

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4751>.

C#

```
(Form1.cs)  
// Create an array containing string values.  
string[] array = new string[] { "AAA", "BBB", "CCC", "DDD" };  
// Import the array into the worksheet and insert it horizontally, starting with the B1 cell.  
worksheet.Import(array, 0, 1, false);  
// Create a two-dimensional array containing string values.  
String[,] names = new String[2, 4]{  
    {"Ann", "Edward", "Angela", "Alex"},  
    {"Rachel", "Bruce", "Barbara", "George"}  
};  
// Import the two-dimensional array into the worksheet and insert it, starting with the B3 cell.  
worksheet.Import(names, 2, 1);
```

Visual Basic

```
(Form1.vb)  
' Create an array containing string values.  
Dim array() As String = { "AAA", "BBB", "CCC", "DDD" }  
' Import the array into the worksheet and insert it horizontally, starting with the B1 cell.  
worksheet.Import(array, 0, 1, False)  
' Create a two-dimensional array containing string values.  
Dim names(,) As String = {  
    {"Ann", "Edward", "Angela", "Alex"},  
    {"Rachel", "Bruce", "Barbara", "George"}  
}  
' Import the two-dimensional array into the worksheet and insert it, starting with the B3 cell.  
worksheet.Import(names, 2, 1)
```

The image below shows the results.

	A	B	C	D	E
1	Import an array horizontally:	AAA	BBB	CCC	DDD
2					
3	Import a two-dimensional array:	Ann	Edward	Angela	Alex
4		Rachel	Bruce	Barbara	George
5					
6					

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of strings.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal array As String[],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    String[] array,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of strings that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array of strings and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from a two-dimensional array of objects.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Object[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal converter As IDataValueConverter  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Object[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex,  
    IDataValueConverter converter  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of objects that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

converter

An object that implements the DevExpress.Spreadsheet.IDataValueConverter interface for data conversion.

Remarks

The **Import** imports data from a two-dimensional array and enables you to process data before insertion in a worksheet.

See Also

[WorksheetExtensions Class](#)
[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from an one-dimensional array of long integers.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(
    ByVal sheet As Worksheet,
    ByVal array As Int64[],
    ByVal firstRowIndex As Integer,
    ByVal firstColumnIndex As Integer,
    ByVal isVertical As Boolean
)
```

C#

```
public static void Import(
    Worksheet sheet,
    Int64[] array,
    int firstRowIndex,
    int firstColumnIndex,
    bool isVertical
)
```

Parameters*sheet*

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of long integers that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of decimal values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(
    ByVal sheet As Worksheet,
    ByVal array As Decimal[],
    ByVal firstRowIndex As Integer,
    ByVal firstColumnIndex As Integer,
    ByVal isVertical As Boolean
)
```

C#

```
public static void Import(
    Worksheet sheet,
    Decimal[] array,
    int firstRowIndex,
    int firstColumnIndex,
    bool isVertical
)
```

Parameters*sheet*

A `DevExpress.Spreadsheet.Worksheet` that is the worksheet to which the data is imported.

array

A one-dimensional array of decimal values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of single-precision floating point values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal array As Single[],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Single[] array,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical  
)
```

Parameters

sheet

A `DevExpress.Spreadsheet.Worksheet` that is the worksheet to which the data is imported.

array

A one-dimensional array of single-precision floating point values that is the data source.

firstRowIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of DateTime values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal array As DateTime[],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    DateTime[] array,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of [System.DateTime](#) values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of strings.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As String[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal options As DataImportOptions  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    String[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex,  
    DataImportOptions options  
)
```

Parameters

- sheet*
A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.
- twoDimensionalArray*
A two-dimensional array of strings that is the data source.
- firstRowIndex*
An integer that is the row index of the start cell in which the imported data will be inserted.
- firstColumnIndex*
An integer that is the column index of the start cell in which the imported data will be inserted.
- options*
A DevExpress.Spreadsheet.DataImportOptions object containing data import options, parameters and converter.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4751>.

C#

```
(Form1.cs)  
string[] arrayR1C1 = new string[] { "a", "b", "=R1C1&R1C2" };  
worksheet.Import(arrayR1C1, 0, 0, false, new DataImportOptions() { ImportFormulas = true, ReferenceStyle = ...  
string[] arrayLocalized = new string[] { "a", "=1,2+1" };  
worksheet.Import(arrayLocalized, 1, 0, false, new DataImportOptions() { ImportFormulas = true, FormulaCulture = new System.Globalization.CultureInfo
```

Visual Basic

```
(Form1.vb)  
Dim arrayR1C1() As String = { "a", "b", "=R1C1&R1C2" }  
worksheet.Import(arrayR1C1, 0, 0, False, New DataImportOptions() With {.ImportFormulas = True, .ReferenceStyle = ...  
Dim arrayLocalized() As String = { "a", "=1,2+1" }  
worksheet.Import(arrayLocalized, 1, 0, False, New DataImportOptions() With {.ImportFormulas = True, .FormulaCulture = ...
```

See Also

- [WorksheetExtensions Class](#)
- [WorksheetExtensions Members](#)
- [DevExpress.Spreadsheet Namespace](#)
- [WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a two-dimensional array of objects.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal twoDimensionalArray As Object[,],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal options As DataImportOptions  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Object[,] twoDimensionalArray,  
    int firstRowIndex,  
    int firstColumnIndex,  
    DataImportOptions options  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

twoDimensionalArray

A two-dimensional array of objects that is the source of data.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

options

A DevExpress.Spreadsheet.DataImportOptions object containing data import options, parameters and converter.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a data source.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal dataSource As Object,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal options As DataSourceImportOptions  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    object dataSource,  
    int firstRowIndex,  
    int firstColumnIndex,  
    DataSourceImportOptions options  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

dataSource

An object that is the data source for import.
firstRowIndex
An integer that is the row index of the start cell in which the imported data will be inserted.
firstColumnIndex
An integer that is the column index of the start cell in which the imported data will be inserted.
options
A DevExpress.Spreadsheet.DataSourceImportOptions object specifying filed names to import.

Remarks

You can specify object properties to import, as illustrated in the following code snippet..

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4751>.

C#

```
(Form1.cs)
List<TestObject> list = new List<TestObject>();
list.Add(new TestObject(1, "1", true));
list.Add(new TestObject(2, "2", false));
worksheet.Import(list, 0, 0, new DataSourceImportOptions() { PropertyNames = new string[] { "BoolValue", "
```

C#

```
(TestObject.cs)
class TestObject
{
    public TestObject(int intValue, string value, bool boolValue)
    {
        this.intValue = intValue;
        this.Value = value;
        this.BoolValue = boolValue;
    }
    public int intValue;
    private int privateValue { get { return 123; } }
    public int IntValue { get { return intValue + privateValue - 123; } }
    public string Value { get; set; }
    public bool BoolValue { get; set; }
    public int this[int index] { get { return index; } }
}
```

Visual Basic

```
(Form1.vb)
Dim list As New List(Of TestObject)()
list.Add(New TestObject(1, "1", True))
list.Add(New TestObject(2, "2", False))
worksheet.Import(list, 0, 0, New DataSourceImportOptions() With {
    .PropertyNames = New String() { "BoolValue", "IntValue" }
})
```

Visual Basic

```

(TestObject.vb)
Friend Class TestObject
    Public Sub New(ByVal intValue As Integer, ByVal value As String, ByVal
        Me.intValue_Renamed = intValue
        Me.Value = value
        Me.BoolValue = boolValue
    End Sub
    Public intValue_Renamed As Integer
    Private ReadOnly Property privateValue() As Integer
        Get
            Return 123
        End Get
    End Property
    Public ReadOnly Property IntValue() As Integer
        Get
            Return intValue_Renamed + privateValue - 123
        End Get
    End Property
    Public Property Value() As String
    Public Property BoolValue() As Boolean
    Default Public ReadOnly Property Item(ByVal index As Integer) As Integer
        Get
            Return index
        End Get
    End Property
End Class

```

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of short integers.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```

static Public Sub Import(
    ByVal sheet As Worksheet,
    ByVal array As Int16[],
    ByVal firstRowIndex As Integer,
    ByVal firstColumnIndex As Integer,
    ByVal isVertical As Boolean
)

```

C#

```
public static void Import(
    Worksheet sheet,
    Int16[] array,
    int firstRowIndex,
    int firstColumnIndex,
    bool isVertical
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of short integers that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of double-precision floating point values.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(
    ByVal sheet As Worksheet,
    ByVal array As Double[],
    ByVal firstRowIndex As Integer,
    ByVal firstColumnIndex As Integer,
    ByVal isVertical As Boolean
)
```

C#

```
public static void Import(
    Worksheet sheet,
    Double[] array,
    int firstRowIndex,
    int firstColumnIndex,
    bool isVertical
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of double-precision floating point values that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of integers.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal array As Int32[],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Int32[] array,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of integers that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

Remarks

The **Import** imports data from a one-dimensional array of integers and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a relational database.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal source As IDataReader,  
    ByVal addHeader As Boolean,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal converter As IDataValueConverter  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    IDataReader source,  
    bool addHeader,  
    int firstRowIndex,  
    int firstColumnIndex,  
    IDataValueConverter converter  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

source

An object that exposes the System.Data.IDataReader interface, e.g., an object returned by the SqlCommand.ExecuteReader method.

addHeader

true, to insert column names to the row above the cells containing imported data; otherwise, **false**.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

converter

An object that implements the DevExpress.Spreadsheet.IDataValueConverter interface for data conversion.

Remarks

Use the **Import** method overload to import data from .NET Framework data providers that access relational databases, such as MS SQL Server, and convert data as required before insertion in a worksheet.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of objects.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal array As Object[],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean,  
    ByVal converter As IDataValueConverter  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Object[] array,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical,  
    IDataValueConverter converter  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of objects that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true, to insert imported data vertically; otherwise, **false**

converter

An object that implements the DevExpress.Spreadsheet.IDataValueConverter interface for data conversion.

Remarks

The **Import** imports data from a one-dimensional array and enables you to process data before insertion in a worksheet and to specify how to place the data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from an one-dimensional array of objects.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal array As Object[],  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean,  
    ByVal options As DataImportOptions  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    Object[] array,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical,  
    DataImportOptions options  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

array

A one-dimensional array of objects that is the data source.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true to insert imported data vertically; otherwise, **false**

options

A DevExpress.Spreadsheet.DataImportOptions object containing data import options, parameters and converter.

Remarks

The **Import** imports data from a one-dimensional array and enables you to specify how to place array data in cells - vertically or horizontally.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a data table.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal source As DataTable,  
    ByVal addHeader As Boolean,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal converter As IDataValueConverter  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    DataTable source,  
    bool addHeader,  
    int firstRowIndex,  
    int firstColumnIndex,  
    IDataValueConverter converter  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

source

A [System.Data.DataTable](#) object that is the data source for import.

addHeader

true, to insert column names to the row above the cells containing imported data; otherwise, **false**.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

converter

An object that implements the `DevExpress.Spreadsheet.IDataValueConverter` interface for data conversion.

Remarks

The **Import** imports data from a data table and enables you to process data before insertion in a worksheet.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a data table.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal source As DataTable,  
    ByVal addHeader As Boolean,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal options As DataImportOptions  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    DataTable source,  
    bool addHeader,  
    int firstRowIndex,  
    int firstColumnIndex,  
    DataImportOptions options  
)
```

Parameters*sheet*

A `DevExpress.Spreadsheet.Worksheet` that is the worksheet to which the data is imported.

source

A [System.Data.DataTable](#) object that is the data source for import.

addHeader

true, to insert column names to the row above the cells containing imported data; otherwise, **false**.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

options

A `DevExpress.Spreadsheet.DataImportOptions` object containing data import options, parameters and converter.

Remarks

The **Import** imports data from a data table and enables you to process data before insertion in a worksheet.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)
[DevExpress.Spreadsheet Namespace](#)
[WorksheetExtensions.Import Overload List](#)
How to: Import Data to a Worksheet

Imports data from a collection.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal source As IEnumerable,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean,  
    ByVal options As DataImportOptions  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    IEnumerable source,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical,  
    DataImportOptions options  
)
```

Parameters

- sheet*
A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.
- source*
An object that exposes the [System.Collections.IEnumerable](#) interface provided by a collection of objects being imported.
- firstRowIndex*
An integer that is the row index of the start cell in which the imported data will be inserted.
- firstColumnIndex*
An integer that is the column index of the start cell in which the imported data will be inserted.
- isVertical*
true, to insert imported data vertically; otherwise, **false**
- options*
A DevExpress.Spreadsheet.DataImportOptions object containing data import options, parameters and converter.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4751>.

C#

(Form1.cs)
List<TestObject> list = new List<TestObject>();
list.Add(new TestObject(1, "1", true));
list.Add(new TestObject(2, "2", false));
worksheet.Import(list, 0, 0, new DataSourceImportOptions() { Converter = new TestDataValueConverter() });

C#

```
(TestDataValueConverter.cs)
class TestDataValueConverter : DevExpress.Spreadsheet.IDataValueConverter
{
    public bool TryConvert(object value, int columnIndex, out DevExpress.Spreadsheet.CellValue result)
    {
        string strValue = value as string;
        if (strValue != null)
        {
            int str2int;
            bool success = Int32.TryParse(strValue, out str2int);
            result = success ? str2int : 0;
            return true;
        }
        Type valueType = value.GetType();
        if (valueType == typeof(int))
            result = (int)value;
        else
            result = null;
        return true;
    }
}
```

C#

```
(TestObject.cs)
class TestObject
{
    public TestObject(int intValue, string value, bool boolValue)
    {
        this.intValue = intValue;
        this.Value = value;
        this.BoolValue = boolValue;
    }
    public int intValue;
    private int privateValue { get { return 123; } }
    public int IntValue { get { return intValue + privateValue - 123; } }
    public string Value { get; set; }
    public bool BoolValue { get; set; }
    public int this[int index] { get { return index; } }
}
```

Visual Basic

```
(Form1.vb)
Dim list As New List(Of TestObject)()
list.Add(New TestObject(1, "1", True))
list.Add(New TestObject(2, "2", False))
worksheet.Import(list, 0, 0, New DataSourceImportOptions() With {.Convert...
```

Visual Basic

```

(TestDataValueConverter.vb)
Friend Class TestDataValueConverter
    Implements DevExpress.Spreadsheet.IDataValueConverter
    Public Function TryConvert(ByVal value As Object, ByVal columnIndex As Integer,
        Dim strValue As String = TryCast(value, String)
        If strValue IsNot Nothing Then
            Dim str2int As Integer = Nothing
            Dim success As Boolean = Int32.TryParse(strValue, str2int)
            result = If(success, str2int, 0)
            Return True
        End If
        Dim valueType As Type = value.GetType()
        If valueType Is GetType(Integer) Then
            result = DirectCast(value, Integer)
        Else
            result = Nothing
        End If
        Return True
    End Function
End Class

```

Visual Basic

```

(TestObject.vb)
Friend Class TestObject
    Public Sub New(ByVal intValue As Integer, ByVal value As String, ByVal boolValue As Boolean)
        Me.intValue_Renamed = intValue
        Me.Value = value
        Me.BoolValue = boolValue
    End Sub
    Public intValue_Renamed As Integer
    Private ReadOnly Property privateValue() As Integer
        Get
            Return 123
        End Get
    End Property
    Public ReadOnly Property IntValue() As Integer
        Get
            Return intValue_Renamed + privateValue - 123
        End Get
    End Property
    Public Property Value() As String
    Public Property BoolValue() As Boolean
    Default Public ReadOnly Property Item(ByVal index As Integer) As Integer
        Get
            Return index
        End Get
    End Property
End Class

```

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#) [WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a relational database.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal source As IDataReader,  
    ByVal addHeader As Boolean,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal options As DataImportOptions  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    IDataReader source,  
    bool addHeader,  
    int firstRowIndex,  
    int firstColumnIndex,  
    DataImportOptions options  
)
```

Parameters

sheet

A DevExpress.Spreadsheet.Worksheet that is the worksheet to which the data is imported.

source

An object that exposes the System.Data.IDataReader interface, e.g., an object returned by the SqlCommand.ExecuteReader method.

addHeader

true, to insert column names to the row above the cells containing imported data; otherwise, **false**.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

options

A DevExpress.Spreadsheet.DataImportOptions object containing data import options, parameters and converter.

Remarks

Use the **Import** method overload to import data from .NET Framework data providers that access relational databases, such as MS SQL Server, and convert data as required before insertion in a worksheet.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

Imports data from a collection.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Import(  
    ByVal sheet As Worksheet,  
    ByVal source As IEnumerable,  
    ByVal firstRowIndex As Integer,  
    ByVal firstColumnIndex As Integer,  
    ByVal isVertical As Boolean,  
    ByVal converter As IDataValueConverter  
)
```

C#

```
public static void Import(  
    Worksheet sheet,  
    IEnumerable source,  
    int firstRowIndex,  
    int firstColumnIndex,  
    bool isVertical,  
    IDataValueConverter converter  
)
```

Parameters

sheet

A `DevExpress.Spreadsheet.Worksheet` that is the worksheet to which the data is imported.

source

An object that exposes the [System.Collections.IEnumerable](#) interface provided by a collection of objects being imported.

firstRowIndex

An integer that is the row index of the start cell in which the imported data will be inserted.

firstColumnIndex

An integer that is the column index of the start cell in which the imported data will be inserted.

isVertical

true, to insert imported data vertically; otherwise, **false**

converter

An object that implements the `DevExpress.Spreadsheet.IDataValueConverter` interface for data conversion.

Remarks

The **Import** imports data from a collection and enables you to process data before insertion in a worksheet.

See Also

[WorksheetExtensions Class](#)

[WorksheetExtensions Members](#)

[DevExpress.Spreadsheet Namespace](#)

[WorksheetExtensions.Import Overload List](#)

How to: Import Data to a Worksheet

DevExpress.Spreadsheet.Export

Contains base interfaces and classes required for data export from the cells of the spreadsheet to a data table.

Classes

	Class	Description
	DataTableExporterExtensions	Defines extension methods for the DevExpress.Spreadsheet.Export.DataTableExporter class.

DataTableExporterExtensions Class

Defines extension methods for the DevExpress.Spreadsheet.Export.DataTableExporter class.

Namespace: [DevExpress.Spreadsheet.Export](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

NotInheritable Public Class DataTableExporterExtensions Inherits [Object](#)

C#

```
public sealed abstract class DataTableExporterExtensions : object
```

Remarks

To enable extensions, add the a reference to the DevExpress.Docs.v18.1.dll assembly and explicitly import the [DevExpress.Spreadsheet.Export](#) namespace into the code with a **using** directive (**Imports** in Visual Basic).

Subsequently, all extension methods become accessible and you can call them using the common instance method syntax.

Use of these methods in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Refer to the [DevExpress Subscription](#) page for pricing information.

Inheritance Hierarchy

[Object](#)

DataTableExporterExtensions

See Also

[DataTableExporterExtensions Members](#)








[DevExpress.Spreadsheet.Export Namespace](#)

DataTableExporterExtensions Members

Defines extension methods for the `DevExpress.Spreadsheet.Export.DataTableExporter` class.

The following tables list the members exposed by the [DataTableExporterExtensions](#) type.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Export	Performs export of the cell range to the DataTable. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[DataTableExporterExtensions Members](#)








[DevExpress.Spreadsheet.Export Namespace](#)

DataTableExporterExtensions Methods

Defines extension methods for the `DevExpress.Spreadsheet.Export.DataTableExporter` class.

The following tables list the members exposed by the [DataTableExporterExtensions](#) type.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Export	Performs export of the cell range to the DataTable. Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[DataTableExporterExtensions Members](#)

[DevExpress.Spreadsheet.Export Namespace](#)

Export Method

Performs export of the cell range to the DataTable.

Use of this method in production code requires a license to the DevExpress Office File API or the DevExpress Universal Subscription.

Namespace: [DevExpress.Spreadsheet.Export](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public Sub Export(  
    ByVal exporter As DataTableExporter  
)
```

C#

```
public static void Export(  
    DataTableExporter exporter  
)
```

Parameters

exporter
A DevExpress.Spreadsheet.Export.DataTableExporter object created to export data.

Example

Show Me

A complete sample project is available in the DevExpress Code Examples database at <http://www.devexpress.com/example=E4997>.

	A	B	C	D	E	F	G	H	I
4		This report shows trade data on the United States on a country by country basis. All values are in million USD.							
5		Country	Exports	Imports	Balance	Exports 1Y	Imports 1Y	Balance 1Y	As Of
6		Australia	2,277.90	815.40	▲ 1,462.50	-8.94%	-2.79%	-0.12%	Jul-13
7		Belgium	2,808.90	1,826.20	▲ 982.70	19.23%	22.91%	0.13%	May-13
8		Brazil	3,423.20	2,571.40	▲ 851.80	-4.64%	18.83%	-0.40%	May-13
9		Canada	26,453.10	28,343.10	▼ -1,890.00	18.87%	9.94%	-0.46%	May-13
10		China	8,786.50						
11		France	2,642.30						
12		Germany	4,052.50						
13		Hong Kong	3,385.40						
14		India	1,935.80						
		Country	Exports	Imports	Balance	Exports 1...	Imports 1...	Balance 1...	As Of
		Australia	2277.9	815.4	1462.5	-0.0894	-0.0279	-0.0012	7/1/2013
		Belgium	2808.9	1826.2	982.7	0.1923	0.2291	0.0013	5/1/2013
		Brazil	3423.2	2571.4	851.8	-0.0464	0.1883	-0.004	5/1/2013
		Canada	26453.1	28343.1	-1890	0.1887	0.0994	-0.0046	5/1/2013
		China	8786.5	36646.2	-27859.7	-0.1478	0.0518	0.0014	5/1/2013
		France	2642.3	3563.7	-921.4	0.0882	0.063	0	5/1/2013
		Germany	4052.5	9888.3	-5835.8	0.0662	0.052	0.0004	5/1/2013
		Hong Kong	3385.4	413	2972.4	-0.2327	0.0034	-0.0026	
		India	1935.8	4201.5	-2265.7	-0.1478	0.0518	0.0014	5/1/2013

C#

```

(Form1.cs)
using DevExpress.Spreadsheet;
using DevExpress.Spreadsheet.Export;

Worksheet worksheet = spreadsheetControl1.Document.Worksheets.ActiveWorksheet;
Range range = worksheet.Selection;
bool rangeHasHeaders = this.barCheckItemHasHeaders1.Checked;
// Create a data table with column names obtained from the first row in a range if it has headers
// Column data types are obtained from cell value types of cells in the first data row of the
DataTable dataTable = worksheet.CreateDataTable(range, rangeHasHeaders);
//Validate cell value types. If cell value types in a column are different, the column values
for (int col = 0; col < range.ColumnCount; col++)
{
    CellValueType cellType = range[0, col].Value.Type;
    for (int r = 1; r < range.RowCount; r++)
    {
        if (cellType != range[r, col].Value.Type)
        {
            dataTable.Columns[col].DataType = typeof(string);
            break;
        }
    }
}
// Create the exporter that obtains data from the specified range,
// skips the header row (if required) and populates the previously created data table.
DataTableExporter exporter = worksheet.CreateDataTableExporter(range, dataTable, rangeHasHeaders);
// Handle value conversion errors.
exporter.CellValueConversionError += exporter_CellValueConversionError;
// Perform the export.
exporter.Export();
void exporter_CellValueConversionError(object sender, CellValueConversionEventArgs e)
{
    MessageBox.Show("Error in cell " + e.Cell.GetReferenceA1());
    e.DataTableValue = null;
    e.Action = DataTableExporterAction.Continue;
}

```

Visual Basic

```

(Form1.vb)
Imports DevExpress.Spreadsheet
Imports DevExpress.Spreadsheet.Export

    Dim worksheet As Worksheet = spreadsheetControll1.Document.Worksheets.ActiveWorksheet
    Dim range As Range = worksheet.Selection
    Dim rangeHasHeaders As Boolean = Me.barCheckItemHasHeaders1.Checked
    ' Create a data table with column names obtained from the first row in a range if it has headers
    ' Column data types are obtained from cell value types of cells in the first data row of the range
    Dim dataTable As DataTable = worksheet.CreateDataTable(range, rangeHasHeaders)
    ' Validate cell value types. If cell value types in a column are different, the column values are converted to strings
    For col As Integer = 0 To range.ColumnCount - 1
        Dim cellType As CellValueType = range(0, col).Value.Type
        For r As Integer = 1 To range.RowCount - 1
            If cellType <> range(r, col).Value.Type Then
                dataTable.Columns(col).DataType = GetType(String)
                Exit For
            End If
        Next r
    Next col
    ' Create the exporter that obtains data from the specified range,
    ' skips the header row (if required) and populates the previously created data table.
    Dim exporter As DataTableExporter = worksheet.CreateDataTableExporter(range, dataTable, rangeHasHeaders)
    ' Handle value conversion errors.
    AddHandler exporter.CellValueConversionError, AddressOf exporter_CellValueConversionError
    ' Perform the export.
    exporter.Export()
    Private Sub exporter_CellValueConversionError(ByVal sender As Object, ByVal e As CellValueConversionError)
        MessageBox.Show("Error in cell " & e.Cell.GetReferenceA1())
        e.DataTableValue = Nothing
        e.Action = DataTableExporterAction.Continue
    End Sub

```

See Also

[DataTableExporterExtensions Class](#)

[DataTableExporterExtensions Members](#)

[DevExpress.Spreadsheet.Export Namespace](#)

DevExpress.UnitConversion

Contains classes of the Unit Conversion library that are essential for the conversion between different units of measurement.

Classes

	Class	Description
	AreaUnitsConverter	Converts area measurement from one unit to another.
	BaseUnitsConverter<T>	Base class for unit converters.
	BinaryUnitsConverter	Converts a value from one binary prefix multiplier to another.
	DistanceUnitsConverter	Converts distance measurement from one unit to another.
	EnergyUnitsConverter	Converts energy measurement from one unit to another.
	ForceUnitsConverter	Converts force measurement from one unit to another.
	InformationUnitsConverter	Converts information measurement from one unit to another.
	MagnetismUnitsConverter	Converts the measurement of a magnetic field from one unit to another.
	MassUnitsConverter	Converts mass measurement from one unit to another.
	MetricUnitsConverter	Converts a value from one metric prefix multiplier to another.
	PowerUnitsConverter	Converts power measurement from one unit to another.
	PrefixUnitsConverter<T>	Base class for prefix converters.
	PressureUnitsConverter	Converts area measurement from one unit to another.
	SpeedUnitsConverter	Converts speed measurement from one unit to another.
	TemperatureUnitsConverter	Converts temperature measurement from one unit to another.
	TimeUnitsConverter	Converts time measurement from one unit to another.
	Units	Provides access to unit converters.
	VolumeUnitsConverter	Converts volume measurement from one unit to another.

Structures

	Structure	Description
--	-----------	-------------

	QuantityValue<T>	Contains the unit of measurement and its value.
--	--	---

Enumerations

	Enumeration	Description
	Area	Lists units of area measurement.
	BinaryPrefix	Lists binary metric prefixes.
	Distance	Lists units of distance measurement.
	Energy	Lists units of energy measurement.
	Force	Lists units of force measurement.
	Information	Lists units of information.
	Magnetism	Lists units of measurement of a magnetic field.
	Mass	Lists units of mass measurement.
	MetricPrefix	Lists metric prefixes.
	Power	Lists units of power measurement.
	Pressure	Lists units of pressure measurement.
	Speed	Lists units of speed measurement.
	Temperature	Lists units of temperature measurement.
	Time	Lists units of time measurement.
	Volume	Lists units of volume measurement.

Area Enumeration

Lists units of area measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Area`
C#
`public enum Area`

Members

Name	Description
AcreInternational	International acre ("uk_acre").
AcreStatute	U.S. survey/statute acre ("us_acre").
Are	Are ("ar").
Hectare	Hectare ("ha").
Morgen	Morgen ("Morgen").
SquareAngstrom	Square angstrom ("ang2").
SquareFoot	Square feet ("ft2").
SquareInch	Square inches ("in2").
SquareLightYear	Square light-year ("ly2").
SquareMeter	Square meters ("m2").
SquareMile	Square miles ("mi2").
SquareMileNautical	Square nautical miles ("Nmi2").
SquarePicaPoint	Square Pica ("Pica2").
SquareYard	Square yards ("yd2").

See Also
[DevExpress.UnitConversion Namespace](#)

AreaUnitsConverter Class

Converts area measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class AreaUnitsConverter Inherits [BaseUnitsConverter\(of Area\)](#)

C#

public class AreaUnitsConverter : [BaseUnitsConverter<Area>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 hectare to square meters, use the following code:

C#	
<pre>double result = Units.Area.Convert(100.0f, Area.Hectare, Area.SquareMeter);</pre>	

Visual Basic

<pre>Dim result As Double = Units.Area.Convert(100.0f, Area.Hectare, Area.SquareMeter)</pre>	

Inheritance Hierarchy

[Object](#)
 [BaseUnitsConverter<Area>](#)
 AreaUnitsConverter

See Also

[AreaUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Area](#)

AreaUnitsConverter Members









Converts area measurement from one unit to another.

The following tables list the members exposed by the [AreaUnitsConverter](#) type.

Public Constructors

	Name	Description
	AreaUnitsConverter	Initializes a new instance of the AreaUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[AreaUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

[Area](#)

AreaUnitsConverter Constructor

Initializes a new instance of the [AreaUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public AreaUnitsConverter()
```

See Also

[AreaUnitsConverter Class](#)

[AreaUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

BaseUnitsConverter<T> Class

Base class for unit converters.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class BaseUnitsConverter(of T) Inherits [Object](#)

C#

public abstract class BaseUnitsConverter<T> : [object](#)

Inheritance Hierarchy

[Object](#)

BaseUnitsConverter<T>

Derived classes

See Also

[BaseUnitsConverter<T> Members](#)









[DevExpress.UnitConversion Namespace](#)

BaseUnitsConverter<T> Members

Base class for unit converters.

The following tables list the members exposed by the [BaseUnitsConverter<T>](#) type.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)









See Also
[BaseUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)

BaseUnitsConverter<T> Methods

Base class for unit converters.

The following tables list the members exposed by the [BaseUnitsConverter<T>](#) type.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[BaseUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)

Convert Method

Converts a value of type Double from one unit of measurement to another.

Overload List

Name	Description
Double Convert(Double value, T from, T to)	Converts a value of type Double from one unit of measurement to another.
void Convert(Double[] values, T from, T to)	Converts an array of type Double from one unit of measurement to another.
void Convert(IList<Double> values, T from, T to)	Converts a list of Double values from one unit of measurement to another.

See Also
[BaseUnitsConverter<T> Class](#)
[BaseUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)
[BaseUnitsConverter<T>.Convert Overload List](#)

Converts a value of type Double from one unit of measurement to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Convert(  
    ByVal value As Double,  
    ByVal from As T,  
    ByVal to As T  
) As Double
```

C#

```
public Double Convert(  
    Double value,  
    T from,  
    T to  
)
```

Parameters

value
A [System.Double](#) source value.
from
Measurement unit of the source value.
to
Measurement unit of the target value.

Return Value

A [System.Double](#) value.

See Also
[BaseUnitsConverter<T> Class](#)
[BaseUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)
[BaseUnitsConverter<T>.Convert Overload List](#)

Converts an array of type Double from one unit of measurement to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub Convert(  
    ByVal values As Double[],  
    ByVal from As T,  
    ByVal to As T  
)
```

C#

```
public void Convert(  
    Double[] values,  
    T from,  
    T to  
)
```

Parameters

values

An array of [System.Double](#) values.

from

Measurement unit of the source value.

to

Measurement unit of the target value.

See Also

[BaseUnitsConverter<T> Class](#)

[BaseUnitsConverter<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

[BaseUnitsConverter<T>.Convert Overload List](#)

Converts a list of Double values from one unit of measurement to another.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Sub Convert(  
    ByVal values As IList(of Double),  
    ByVal from As T,  
    ByVal to As T  
)
```

C#

```
public void Convert(  
    IList<Double> values,  
    T from,  
    T to  
)
```

Parameters

values

An System.Collections.Generic.IList<System.Double>array of values.

from

Measurement unit of the source value.

to

Measurement unit of the target value.

See Also

[BaseUnitsConverter<T> Class](#)

[BaseUnitsConverter<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

[BaseUnitsConverter<T>.Convert Overload List](#)

GetTransform Method

Obtains a delegate function that performs the conversion.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetTransform(  
    ByVal from As T,  
    ByVal to As T  
) As Func(of Double, Double)
```

C#

```
public Func<Double, Double> GetTransform(  
    T from,  
    T to  
)
```

Parameters

from

Measurement unit of the source value.

to

Measurement unit of the target value.

Return Value

A delegate method that performs the conversion.

See Also

[BaseUnitsConverter<T> Class](#)

[BaseUnitsConverter<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

BinaryPrefix Enumeration

Lists binary metric prefixes.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum BinaryPrefix`
C#
`public enum BinaryPrefix`

Members

Name	Description
Exa	$2^{60} = 1\,152\,921\,504\,606\,846\,976$ ("Ei")
Exbi	$2^{60} = 1\,152\,921\,504\,606\,846\,976$ ("Ei")
Gibi	$2^{30} = 1\,073\,741\,824$ ("Gi")
Giga	$2^{30} = 1\,073\,741\,824$ ("Gi")
Kibi	$2^{10} = 1024$ ("ki")
Kilo	$2^{10} = 1024$ ("ki")
Mebi	$2^{20} = 1\,048\,576$ "Mi"
Mega	$2^{20} = 1\,048\,576$ "Mi"
None	1
Pebi	$2^{50} = 1\,125\,899\,906\,842\,624$ ("Pi")
Peta	$2^{50} = 1\,125\,899\,906\,842\,624$ ("Pi")
Tebi	$2^{40} = 1\,099\,511\,627\,776$ ("Ti")
Tera	$2^{40} = 1\,099\,511\,627\,776$ ("Ti")
Yobi	$2^{80} = 1\,208\,925\,819\,614\,629\,174\,706\,176$ ("Yi")
Yotta	$2^{80} = 1\,208\,925\,819\,614\,629\,174\,706\,176$ ("Yi")
Zebi	$2^{70} = 1\,180\,591\,620\,717\,411\,303\,424$ ("Zi")
Zetta	$2^{70} = 1\,180\,591\,620\,717\,411\,303\,424$ ("Zi")

See Also
[DevExpress.UnitConversion Namespace](#)

BinaryUnitsConverter Class

Converts a value from one binary prefix multiplier to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class BinaryUnitsConverter Inherits [PrefixUnitsConverter\(of BinaryPrefix\)](#)

C#

public class BinaryUnitsConverter : [PrefixUnitsConverter<BinaryPrefix>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform conversion.

To calculate how many bytes in a tebibyte, use the following code:

C#	
<code>double result = Units.Binary.Convert(1.0f, BinaryPrefix.Tebi, BinaryPrefix.None);</code>	

Visual Basic

<code>Dim result As Double = Units.Binary.Convert(1.0F, BinaryPrefix.Tebi, BinaryPrefix.None)</code>	

Inheritance Hierarchy



See Also


- [BinaryUnitsConverter Members](#)
- [DevExpress.UnitConversion Namespace](#)
- [BinaryPrefix](#)

BinaryUnitsConverter Members







Converts a value from one binary prefix multiplier to another.

The following tables list the members exposed by the [BinaryUnitsConverter](#) type.

Public Constructors

	Name	Description
	BinaryUnitsConverter	Initializes a new instance of the BinaryUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Performs a metric conversion of a list of Double values from one metric prefix to another. (Inherited from PrefixUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from PrefixUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[BinaryUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[BinaryPrefix](#)

BinaryUnitsConverter Constructor

Initializes a new instance of the [BinaryUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public BinaryUnitsConverter()
```

See Also

[BinaryUnitsConverter Class](#)

[BinaryUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Distance Enumeration

Lists units of distance measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Distance`
C#
`public enum Distance`

Members

Name	Description
Angstrom	Angstrom ("ang").
Ell	Ell ("ell").
Foot	Foot ("ft").
Inch	Inch ("in").
LightYear	Light-year ("ly").
Meter	Meter ("m").
MileNautical	Nautical mile ("Nmi").
MileStatute	Statute mile ("mi").
MileUSSurvey	U.S survey mile (statute mile) ("survey_mi").
Parsec	Parsec ("parsec").
Pica	Pica (1/6 inch) "pica".
PicaPoint	Pica (1/72 inch) ("Pica").
Yard	Yard ("yd").

See Also
[DevExpress.UnitConversion Namespace](#)

DistanceUnitsConverter Class

Converts distance measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class DistanceUnitsConverter Inherits [BaseUnitsConverter\(of Distance\)](#)

C#

public class DistanceUnitsConverter : [BaseUnitsConverter<Distance>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 feet to meters, use the following code:

C#	
<code>double result = Units.Distance.Convert(100.0f, Distance.Foot, Distance.Meter);</code>	

Visual Basic

<code>Dim result As Double = Units.Distance.Convert(100.0F, Distance.Foot, Distance.Meter)</code>	

Inheritance Hierarchy



See Also


- [DistanceUnitsConverter Members](#)
- [DevExpress.UnitConversion Namespace](#)
- [Distance](#)

DistanceUnitsConverter Members

Converts distance measurement from one unit to another.

The following tables list the members exposed by the [DistanceUnitsConverter](#) type.

Public Constructors

	Name	Description
	DistanceUnitsConverter	Initializes a new instance of the DistanceUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[DistanceUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Distance](#)

DistanceUnitsConverter Constructor

Initializes a new instance of the [DistanceUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public DistanceUnitsConverter()
```

See Also

[DistanceUnitsConverter Class](#)

[DistanceUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Energy Enumeration

Lists units of energy measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Energy`
C#
`public enum Energy`

Members

Name	Description
BritishThermalUnit	British Thermal Unit ("btu").
Btu	British Thermal Unit ("btu").
CalorieIT	IT calorie ("cal").
CalorieThermodynamic	Thermodynamic calorie ("c").
ElectronVolt	Electron volt ("eV").
Erg	Erg ("e").
Ev	Electron volt ("eV").
Flb	Foot-pound ('flb').
FootPound	Foot-pound ('flb').
HorsePowerHour	Horsepower-hour ("HPh").
HPh	Horsepower-hour ("HPh").
Joule	Joule ("J").
WattHour	Watt-hour ("Wh").
Wh	Watt-hour ("Wh").

See Also
[DevExpress.UnitConversion Namespace](#)

EnergyUnitsConverter Class

Converts energy measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class EnergyUnitsConverter Inherits [BaseUnitsConverter\(of Energy\)](#)

C#

public class EnergyUnitsConverter : [BaseUnitsConverter<Energy>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 British Thermal Units to Joules, use the following code:

C#	
<pre>double result = Units.Energy.Convert(100.0f, Energy.Btu, Energy.Joule);</pre>	

Visual Basic

<pre>Dim result As Double = Units.Energy.Convert(100.0F, Energy.Btu, Energy.Joule)</pre>	

Inheritance Hierarchy



See Also


- [EnergyUnitsConverter Members](#)
- [DevExpress.UnitConversion Namespace](#)
- [Energy](#)

EnergyUnitsConverter Members









Converts energy measurement from one unit to another.

The following tables list the members exposed by the [EnergyUnitsConverter](#) type.

Public Constructors

	Name	Description
	EnergyUnitsConverter	Initializes a new instance of the EnergyUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[EnergyUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

[Energy](#)

EnergyUnitsConverter Constructor

Initializes a new instance of the [EnergyUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public EnergyUnitsConverter()
```

See Also

[EnergyUnitsConverter Class](#)

[EnergyUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Force Enumeration

Lists units of force measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum Force
```

C#

```
public enum Force
```

Members

Name	Description
Dyne	Dyne ("dyn").
Lbf	Pound force ("lbf").
Newton	Newton ("N").
Pond	Pond ("pond").
Pound	Pound force ("lbf").

See Also

[DevExpress.UnitConversion Namespace](#)

ForceUnitsConverter Class

Converts force measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class ForceUnitsConverter Inherits [BaseUnitsConverter\(of Force\)](#)

C#

public class ForceUnitsConverter : [BaseUnitsConverter<Force>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 pound-force to Newtons, use the following code:

C#	
<pre>double result = Units.Force.Convert(100.0f, Force.Lbf, Force.Newton);</pre>	

Visual Basic

<pre>Dim result As Double = Units.Force.Convert(100.0F, Force.Lbf, Force.Newton)</pre>	

Inheritance Hierarchy

[Object](#)
 [BaseUnitsConverter<Force>](#)
 ForceUnitsConverter

See Also


[ForceUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Force](#)

ForceUnitsConverter Members







Converts force measurement from one unit to another.

The following tables list the members exposed by the [ForceUnitsConverter](#) type.

Public Constructors

	Name	Description
	ForceUnitsConverter	Initializes a new instance of the ForceUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[ForceUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Force](#)

ForceUnitsConverter Constructor

Initializes a new instance of the [ForceUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public ForceUnitsConverter()
```

See Also

[ForceUnitsConverter Class](#)

[ForceUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Information Enumeration

Lists units of information.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Information`
C#
`public enum Information`

Members

Name	Description
Bit	Bit ("bit").
Byte	Byte ("byte").

See Also
[DevExpress.UnitConversion Namespace](#)

InformationUnitsConverter Class

Converts information measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class InformationUnitsConverter Inherits [BaseUnitsConverter\(of Information\)](#)

C#

```
public class InformationUnitsConverter : BaseUnitsConverter<Information>
```

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

Inheritance Hierarchy

[Object](#)

[BaseUnitsConverter<Information>](#)

InformationUnitsConverter

See Also

[InformationUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)


[Information](#)

InformationUnitsConverter Members






Converts information measurement from one unit to another.

The following tables list the members exposed by the [InformationUnitsConverter](#) type.

Public Constructors

	Name	Description
	InformationUnitsConverter	Initializes a new instance of the InformationUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
 	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
 	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[InformationUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Information](#)

InformationUnitsConverter Constructor

Initializes a new instance of the [InformationUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public InformationUnitsConverter()
```

See Also

[InformationUnitsConverter Class](#)

[InformationUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Magnetism Enumeration

Lists units of measurement of a magnetic field.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Magnetism`
C#
`public enum Magnetism`

Members

Name	Description
Gauss	Gauss ("ga").
Tesla	Tesla ("T").

See Also
[DevExpress.UnitConversion Namespace](#)

MagnetismUnitsConverter Class

Converts the measurement of a magnetic field from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class MagnetismUnitsConverter Inherits [BaseUnitsConverter\(of Magnetism\)](#)

C#

public class MagnetismUnitsConverter : [BaseUnitsConverter<Magnetism>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 Gauss to Tesla, use the following code:

C#	
<code>double result = Units.Magnetism.Convert(100.0f, Magnetism.Gauss, Magnetism.Tesla);</code>	

Visual Basic

<code>Dim result As Double = Units.Magnetism.Convert(100.0F, Magnetism.Gauss, Magnetism.Tesla)</code>	

Inheritance Hierarchy

[Object](#)
 [BaseUnitsConverter<Magnetism>](#)
 MagnetismUnitsConverter

See Also


[MagnetismUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Magnetism](#)

MagnetismUnitsConverter Members









Converts the measurement of a magnetic field from one unit to another.

The following tables list the members exposed by the [MagnetismUnitsConverter](#) type.

Public Constructors

	Name	Description
	MagnetismUnitsConverter	Initializes a new instance of the MagnetismUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[MagnetismUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Magnetism](#)

MagnetismUnitsConverter Constructor

Initializes a new instance of the [MagnetismUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public MagnetismUnitsConverter()
```

See Also

[MagnetismUnitsConverter Class](#)

[MagnetismUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Mass Enumeration

Lists units of mass measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Mass`
C#
`public enum Mass`

Members

Name	Description
AtomicMassUnit	U (atomic mass unit) "u".
Cwt	U.S. (short) hundredweight ("cwt").
Grain	Grain ("grain").
Gram	Gram ("g").
Hundredweight	U.S. (short) hundredweight ("cwt").
HundredweightImperial	Imperial hundredweight ("lcwt").
Lbm	Pound mass (avoirdupois) ("lbm").
Ounce	Ounce mass (avoirdupois) (ozm").
Ozm	Ounce mass (avoirdupois) (ozm").
Pound	Pound mass (avoirdupois) ("lbm").
Slug	Slug ("sg").
Stone	Stone ("stone").
Ton	Ton ("ton").
TonImperial	Imperial ton ("brton").

See Also
[DevExpress.UnitConversion Namespace](#)

MassUnitsConverter Class

Converts mass measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class MassUnitsConverter Inherits [BaseUnitsConverter\(of Mass\)](#)

C#

public class MassUnitsConverter : [BaseUnitsConverter<Mass>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 ounces to grams, use the following code:

C#	
<pre>double result = Units.Mass.Convert(100.0f, Mass.Ounce, Mass.Gram);</pre>	

Visual Basic

<pre>Dim result As Double = Units.Mass.Convert(100.0F, Mass.Ounce, Mass.Gram)</pre>	

Inheritance Hierarchy

[Object](#)
 [BaseUnitsConverter<Mass>](#)
 MassUnitsConverter

See Also

[MassUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Mass](#)

MassUnitsConverter Members









Converts mass measurement from one unit to another.

The following tables list the members exposed by the [MassUnitsConverter](#) type.

Public Constructors

	Name	Description
	MassUnitsConverter	Initializes a new instance of the MassUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[MassUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

[Mass](#)

MassUnitsConverter Constructor

Initializes a new instance of the [MassUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public MassUnitsConverter()
```

See Also

[MassUnitsConverter Class](#)

[MassUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

MetricPrefix Enumeration

Lists metric prefixes.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum MetricPrefix`
C#
`public enum MetricPrefix`

Members

Name	Description
Atto	1E-18 "a"
Centi	1E-02 ("c")
Deci	1E-01 ("d")
Dekao	1E+01 ("e")
Exa	1E+18 ("E")
Femto	1E-15 ("f")
Giga	1E+09 ("G")
Hecto	1E+02 (h")
Kilo	1E+03 ("k")
Mega	1E+06 ("M")
Micro	1E-06 ("u")
Milli	1E-03 ("m")
Nano	1E-09 ("n")
None	1
Peta	1E+15 ("P")
Pico	1E-12 ("p")
Tera	1E+12 ("T")
Yocto	1E-24 ("y")
Yotta	1E+24 ("Y")
Zepto	1E-21 ("z")
Zetta	1E+21 ("Z")

See Also
[DevExpress.UnitConversion Namespace](#)

MetricUnitsConverter Class

Converts a value from one metric prefix multiplier to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class MetricUnitsConverter Inherits [PrefixUnitsConverter\(of MetricPrefix\)](#)

C#

public class MetricUnitsConverter : [PrefixUnitsConverter<MetricPrefix>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To calculate the value of *femto* , use the following code:

C#	
<code>double result = Units.Metric.Convert(1.0f, MetricPrefix.Femto, MetricPrefix.None);</code>	

Visual Basic

<code>Dim result As Double = Units.Metric.Convert(1.0f, MetricPrefix.Femto, MetricPrefix.None)</code>	
---	--

Inheritance Hierarchy



See Also

- [MetricUnitsConverter Members](#)
- [DevExpress.UnitConversion Namespace](#)
- [MetricPrefix](#)

MetricUnitsConverter Members


Converts a value from one metric prefix multiplier to another.

The following tables list the members exposed by the [MetricUnitsConverter](#) type.

Public Constructors

	Name	Description
	MetricUnitsConverter	Initializes a new instance of the MetricUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Performs a metric conversion of a list of Double values from one metric prefix to another. (Inherited from PrefixUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from PrefixUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[MetricUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

[MetricPrefix](#)

MetricUnitsConverter Constructor

Initializes a new instance of the [MetricUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public MetricUnitsConverter()
```

See Also

[MetricUnitsConverter Class](#)

[MetricUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Power Enumeration

Lists units of power measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum Power
```

C#

```
public enum Power
```

Members

Name	Description
HorsePower	Horsepower ("HP").
HP	Horsepower ("HP").
PS	Pferdestarke (HorsePower in German) "PS".
Watt	Watt ("W").

See Also

[DevExpress.UnitConversion Namespace](#)

PowerUnitsConverter Class

Converts power measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class PowerUnitsConverter Inherits [BaseUnitsConverter\(of Power\)](#)

C#

public class PowerUnitsConverter : [BaseUnitsConverter<Power>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 horsepowers to Watts, use the following code:

C#	
<code>double result = Units.Power.Convert(100.0f, Power.HorsePower, Power.Watt);</code>	

Visual Basic

<code>Dim result As Double = Units.Power.Convert(100.0f, Power.HorsePower, Power.Watt)</code>	
---	--

Inheritance Hierarchy



See Also


- [PowerUnitsConverter Members](#)
- [DevExpress.UnitConversion Namespace](#)
- [Power](#)

PowerUnitsConverter Members





Converts power measurement from one unit to another.

The following tables list the members exposed by the [PowerUnitsConverter](#) type.

Public Constructors

	Name	Description
	PowerUnitsConverter	Initializes a new instance of the PowerUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[PowerUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Power](#)

PowerUnitsConverter Constructor

Initializes a new instance of the [PowerUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public PowerUnitsConverter()
```

See Also

[PowerUnitsConverter Class](#)

[PowerUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

PrefixUnitsConverter<T> Class

Base class for prefix converters.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Class PrefixUnitsConverter(of T) Inherits Object
```

C#

```
public abstract class PrefixUnitsConverter<T> : object
```

Inheritance Hierarchy

[Object](#)

PrefixUnitsConverter<T>

Derived classes

See Also

[PrefixUnitsConverter<T> Members](#)









[DevExpress.UnitConversion Namespace](#)

PrefixUnitsConverter<T> Members

Base class for prefix converters.

The following tables list the members exposed by the [PrefixUnitsConverter<T>](#) type.

Public Methods

	Name	Description
	Convert	Overloaded. Performs a metric conversion of a list of Double values from one metric prefix to another.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)









See Also
[PrefixUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)

PrefixUnitsConverter<T> Methods

Base class for prefix converters.

The following tables list the members exposed by the [PrefixUnitsConverter<T>](#) type.

Public Methods

	Name	Description
	Convert	Overloaded. Performs a metric conversion of a list of Double values from one metric prefix to another.
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion.
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[PrefixUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)

Convert Method

Use the Convert(double value, T from, T to) method instead.

Overload List

Name	Description
Double Convert(Double value, T prefix)	Use the Convert(double value, T from, T to) method instead.
void Convert(Double[] values, T prefix)	Use the Convert(double[] values, T from, T to) method instead.
void Convert(ICollection<Double> values, T prefix)	Use the Convert(ICollection<double> values, T from, T to) method instead.
void Convert(ICollection<Double> values, T from, T to)	Performs a metric conversion of a list of Double values from one metric prefix to another.
void Convert(Double[] values, T from, T to)	Performs a metric conversion of an array of Double values from one metric prefix to another.
Double Convert(Double value, T from, T to)	Performs a metric conversion of a Double value from one metric prefix to another.

See Also
[PrefixUnitsConverter<T> Class](#)
[PrefixUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)
[PrefixUnitsConverter<T>.Convert Overload List](#)

NOTE: This member is now obsolete.

Please use the 'void Convert(double value, T from, T to)'

Use the Convert(double value, T from, T to) method instead.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function Convert(  
    ByVal value As Double,  
    ByVal prefix As T  
) As Double
```

C#

```
public Double Convert(  
    Double value,  
    T prefix  
)
```

Parameters

value

prefix

Return Value

See Also
[PrefixUnitsConverter<T> Class](#)
[PrefixUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)
[PrefixUnitsConverter<T>.Convert Overload List](#)

NOTE: This member is now obsolete.

Please use the 'void Convert(double[] values, T from, T to)'

Use the Convert(double[] values, T from, T to) method instead.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Convert(  
    ByVal values As Double[],  
    ByVal prefix As T  
)
```

C#

```
public void Convert(  
    Double[] values,  
    T prefix  
)
```

Parameters

values

prefix

Remarks

\$

See Also

[PrefixUnitsConverter<T> Class](#)

[PrefixUnitsConverter<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

[PrefixUnitsConverter<T>.Convert Overload List](#)

NOTE: This member is now obsolete.

Please use the 'void Convert(IList<double> values, T from, T to)'

Use the Convert(IList<double> values, T from, T to) method instead.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Convert(  
    ByVal values As IList(of Double),  
    ByVal prefix As T  
)
```

C#

```
public void Convert(  
    IList<Double> values,  
    T prefix  
)
```

Parameters

values

prefix

See Also

[PrefixUnitsConverter<T> Class](#)

[PrefixUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)
[PrefixUnitsConverter<T>.Convert Overload List](#)

Performs a metric conversion of a list of Double values from one metric prefix to another.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Convert(  
    ByVal values As IList(of Double),  
    ByVal from As T,  
    ByVal to As T  
)
```

C#

```
public void Convert(  
    IList<Double> values,  
    T from,  
    T to  
)
```

Parameters

values

An System.Collections.Generic.IList<System.Double>array of values.

from

Source metric prefix.

to

Target metric prefix.

See Also

[PrefixUnitsConverter<T> Class](#)
[PrefixUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)
[PrefixUnitsConverter<T>.Convert Overload List](#)

Performs a metric conversion of an array of Double values from one metric prefix to another.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Sub Convert(  
    ByVal values As Double[],  
    ByVal from As T,  
    ByVal to As T  
)
```

C#

```
public void Convert(  
    Double[] values,  
    T from,  
    T to  
)
```

Parameters

values

An array of [System.Double](#) values.

from

Source metric prefix.

to

Target metric prefix.

See Also

[PrefixUnitsConverter<T> Class](#)
[PrefixUnitsConverter<T> Members](#)
[DevExpress.UnitConversion Namespace](#)

[PrefixUnitsConverter<T>.Convert Overload List](#)

Performs a metric conversion of a Double value from one metric prefix to another.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax**Visual Basic**

```
Public Function Convert(  
    ByVal value As Double,  
    ByVal from As T,  
    ByVal to As T  
) As Double
```

C#

```
public Double Convert(  
    Double value,  
    T from,  
    T to  
)
```

Parameters

value

A [System.Double](#) source value.

from

Source metric prefix.

to

Target metric prefix.

Return Value

A [System.Double](#) value.

See Also

[PrefixUnitsConverter<T> Class](#)

[PrefixUnitsConverter<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

[PrefixUnitsConverter<T>.Convert Overload List](#)

GetTransform Method

Obtains a delegate function that performs the conversion.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Function GetTransform(  
    ByVal from As T,  
    ByVal to As T  
) As Func(of Double, Double)
```

C#

```
public Func<Double, Double> GetTransform(  
    T from,  
    T to  
)
```

Parameters

from

Source metric prefix.

to

Target metric prefix.

Return Value

A delegate method that performs the conversion.

See Also

[PrefixUnitsConverter<T> Class](#)

[PrefixUnitsConverter<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

Pressure Enumeration

Lists units of pressure measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Pressure`
C#
`public enum Pressure`

Members

Name	Description
Atm	Atmosphere ("atm").
Atmosphere	Atmosphere ("atm").
MmHg	mm of Mercury ("mmHg").
P	Pascal ("Pa").
Pa	Pascal ("Pa").
Pascal	Pascal ("Pa").
PoundPerSquareInch	Pound-force per square inch ("psi").
Psi	Pound-force per square inch ("psi").
Torr	Torr ("Torr").

See Also
[DevExpress.UnitConversion Namespace](#)

PressureUnitsConverter Class

Converts area measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class PressureUnitsConverter Inherits [BaseUnitsConverter\(of Pressure\)](#)

C#

public class PressureUnitsConverter : [BaseUnitsConverter<Pressure>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 1 atmosphere to millimeters of mercury, use the following code:

C#	
<pre>double result = Units.Pressure.Convert(1.0f, Pressure.Atmosphere, Pressure.MmHg);</pre>	

Visual Basic

<pre>Dim result As Double = Units.Pressure.Convert(1.0f, Pressure.Atmosphere, Pressure.MmHg)</pre>	

Inheritance Hierarchy



See Also

- [PressureUnitsConverter Members](#)
- [DevExpress.UnitConversion Namespace](#)
- [Pressure](#)

PressureUnitsConverter Members

Converts area measurement from one unit to another.

The following tables list the members exposed by the [PressureUnitsConverter](#) type.

Public Constructors

	Name	Description
	PressureUnitsConverter	Initializes a new instance of the PressureUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[PressureUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Pressure](#)

PressureUnitsConverter Constructor

Initializes a new instance of the [PressureUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public PressureUnitsConverter()
```

See Also

[PressureUnitsConverter Class](#)

[PressureUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

QuantityValue<T> Structure

Contains the unit of measurement and its value.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Struct QuantityValue(of T)
```

C#

```
public struct QuantityValue<T>
```

Remarks

[QuantityValue<T>](#) objects are used to perform calculations with physical values.

Inheritance Hierarchy

[Object](#)

System.ValueType

QuantityValue<T>

See Also

[QuantityValue<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

QuantityValue<T> Members







Contains the unit of measurement and its value.

The following tables list the members exposed by the [QuantityValue<T>](#) type.

Public Properties

	Name	Description
	Value	Gets or sets a numerical value of the quantity expressed in the base measurement unit of the specified type.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Indicates whether this instance and a specified object are equal. (Inherited from System.ValueType)
	GetHashCode	Returns the hash code for this instance. (Inherited from System.ValueType)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns the fully qualified type name of this instance. (Inherited from System.ValueType)

See Also

[QuantityValue<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

QuantityValue<T> Properties







Contains the unit of measurement and its value.

The following tables list the members exposed by the [QuantityValue<T>](#) type.

Public Properties

	Name	Description
	Value	Gets or sets a numerical value of the quantity expressed in the base measurement unit of the specified type.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	Equals	Indicates whether this instance and a specified object are equal. (Inherited from System.ValueType)
	GetHashCode	Returns the hash code for this instance. (Inherited from System.ValueType)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns the fully qualified type name of this instance. (Inherited from System.ValueType)

See Also

[QuantityValue<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

Value Property

Gets or sets a numerical value of the quantity expressed in the base measurement unit of the specified type.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Property Value As Double
```

C#

```
public Double Value { get; set; }
```

Property Value

A [System.Double](#) value of the quantity.

Remarks

The **Value** gets or sets a value in the base measurement unit, i.e., the first unit in the inner conversion table. To use the quantity values correctly, set the numerical value using one of the extension functions as illustrated in the following code snippet:

C#

```
using DevExpress.UnitConversion;
//...
// The height is 5'4".
QuantityValue<Distance> height = (5.0).Feet() + (4.0).Inches();
string s = String.Format("The height is {0} ells or {1} meters.",
    height.ToElls().Value.ToString("g3"), height.ToMeters().Value.ToString("g3"));
MessageBox.Show(s);
```

Visual Basic

```
Imports DevExpress.UnitConversion
'...
' The height is 5'4".
Private height As QuantityValue(Of Distance) = (5.0).Feet() + (4.0).Inches()
Private s As String = String.Format("The height is {0} ells or {1} meters.",
    height.ToElls().Value.ToString("g3"), height.ToMeters().Value.ToString("g3"))
MessageBox.Show(s)
```

See Also

[QuantityValue<T> Structure](#)

[QuantityValue<T> Members](#)

[DevExpress.UnitConversion Namespace](#)

Speed Enumeration

Lists units of speed measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum Speed
```

C#

```
public enum Speed
```

Members

Name	Description
Knot	Knot ("kn").
KnotAdmiralty	Admiralty knot ("admkn").
MetersPerHour	Meters per hour ("m/h").
MetersPerSecond	Meters per second ("m/s").
MilesPerHour	Miles per hour ("mph").
Mph	Miles per hour ("mph").

See Also

[DevExpress.UnitConversion Namespace](#)

SpeedUnitsConverter Class

Converts speed measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class SpeedUnitsConverter Inherits [BaseUnitsConverter\(of Speed\)](#)

C#

public class SpeedUnitsConverter : [BaseUnitsConverter<Speed>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert one knot to meters per hour, use the following code:

C#	
<pre>double result = Units.Speed.Convert(1.0f, Speed.Knot, Speed.MetersPerHour);</pre>	

Visual Basic

<pre>Dim result As Double = Units.Speed.Convert(1.0f, Speed.Knot, Speed.MetersPerHour)</pre>	

Inheritance Hierarchy



See Also


- [SpeedUnitsConverter Members](#)
- [DevExpress.UnitConversion Namespace](#)
- [Speed](#)

SpeedUnitsConverter Members





Converts speed measurement from one unit to another.

The following tables list the members exposed by the [SpeedUnitsConverter](#) type.

Public Constructors

	Name	Description
	SpeedUnitsConverter	Initializes a new instance of the SpeedUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[SpeedUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Speed](#)

SpeedUnitsConverter Constructor

Initializes a new instance of the [SpeedUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public SpeedUnitsConverter()
```

See Also

[SpeedUnitsConverter Class](#)

[SpeedUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Temperature Enumeration

Lists units of temperature measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Temperature`
C#
`public enum Temperature`

Members

Name	Description
C	Degree Celsius ("C").
Celcius	Obsolete. Degree Celsius ("C").
Celsius	Degree Celsius ("C").
F	Degree Fahrenheit ("F").
Fahrenheit	Degree Fahrenheit ("F").
K	Kelvin ("K").
Kelvin	Kelvin ("K").
Rankine	Degree Rankine ("Rank").
Reaumur	Degree Reaumur ("Reau").

See Also
[DevExpress.UnitConversion Namespace](#)

TemperatureUnitsConverter Class

Converts temperature measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class TemperatureUnitsConverter Inherits [BaseUnitsConverter\(of Temperature\)](#)

C#

public class TemperatureUnitsConverter : [BaseUnitsConverter<Temperature>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 Celsius to Fahrenheit, use the following code:

C#

```
double result = Units.Temperature.Convert(100.0f, Temperature.Celsius, Temperature.Fahrenheit);
```

Visual Basic

```
Dim result As Double = Units.Temperature.Convert(100.0f, Temperature.Celsius, Temperature.Fahrenheit)
```

Inheritance Hierarchy

Object

[BaseUnitsConverter<Temperature>](#)

TemperatureUnitsConverter

See Also

[TemperatureUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)


[Temperature](#)

TemperatureUnitsConverter Members

Converts temperature measurement from one unit to another.

The following tables list the members exposed by the [TemperatureUnitsConverter](#) type.

Public Constructors

	Name	Description
	TemperatureUnitsConverter	Initializes a new instance of the TemperatureUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also

[TemperatureUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Temperature](#)

TemperatureUnitsConverter Constructor

Initializes a new instance of the [TemperatureUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public TemperatureUnitsConverter()
```

See Also

[TemperatureUnitsConverter Class](#)

[TemperatureUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Time Enumeration

Lists units of time measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public Enum Time
```

C#

```
public enum Time
```

Members

Name	Description
Day	Day ("day").
Hour	Hour ("hr").
Minute	Minute ("min").
Second	Second ("sec").
Year	Year ("yr").

See Also

[DevExpress.UnitConversion Namespace](#)

TimeUnitsConverter Class

Converts time measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class TimeUnitsConverter Inherits [BaseUnitsConverter\(of Time\)](#)

C#

public class TimeUnitsConverter : [BaseUnitsConverter<Time>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert one year to minutes, use the following code:

C#

```
double result = Units.Time.Convert(1.0f, Time.Year, Time.Minute);
```

Visual Basic

```
Dim result As Double = Units.Time.Convert(1.0f, Time.Year, Time.Minute)
```

Inheritance Hierarchy

Object

[BaseUnitsConverter<Time>](#)

TimeUnitsConverter

See Also

[TimeUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)


[Time](#)

TimeUnitsConverter Members









Converts time measurement from one unit to another.

The following tables list the members exposed by the [TimeUnitsConverter](#) type.

Public Constructors

	Name	Description
	TimeUnitsConverter	Initializes a new instance of the TimeUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[TimeUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Time](#)

TimeUnitsConverter Constructor

Initializes a new instance of the [TimeUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public TimeUnitsConverter()
```

See Also

[TimeUnitsConverter Class](#)

[TimeUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)

Units Class

Provides access to unit converters.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

NotInheritable Public Class Units Inherits [Object](#)

C#

```
public sealed abstract class Units : object
```

Inheritance Hierarchy

[Object](#)

Units

See Also

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Units Members


Provides access to unit converters.

The following tables list the members exposed by the [Units](#) type.

Public Properties

	Name	Description
	Area	Provides access to the unit converter for area measurement units.
	Binary	Provides access to unit converter for binary metric prefixes.
	Distance	Provides access to the unit converter for distance measurement units.
	Energy	Provides access to the unit converter for energy measurement units.
	Force	Provides access to the unit converter for force measurement units.
	Information	Provides access to the unit converter for information measurement units.
	Magnetism	Provides access to the unit converter for magnetic field measurement units.
	Mass	Provides access to the unit converter for mass measurement units.
	Metric	Provides access to the unit converter for metric prefixes.
	Power	Provides access to the unit converter for power measurement units.
	Pressure	Provides access to the unit converter for pressure measurement units.
	Speed	Provides access to the unit converter for speed measurement units.
	Temperature	Provides access to the unit converter for temperature measurement units.
	Time	Provides access to the unit converter for time measurement units.
	Volume	Provides access to the unit converter for volume measurement units.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)

	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[Units Members](#)
[DevExpress.UnitConversion Namespace](#)

Units Properties


Provides access to unit converters.

The following tables list the members exposed by the [Units](#) type.

Public Properties

	Name	Description
	Area	Provides access to the unit converter for area measurement units.
	Binary	Provides access to unit converter for binary metric prefixes.
	Distance	Provides access to the unit converter for distance measurement units.
	Energy	Provides access to the unit converter for energy measurement units.
	Force	Provides access to the unit converter for force measurement units.
	Information	Provides access to the unit converter for information measurement units.
	Magnetism	Provides access to the unit converter for magnetic field measurement units.
	Mass	Provides access to the unit converter for mass measurement units.
	Metric	Provides access to the unit converter for metric prefixes.
	Power	Provides access to the unit converter for power measurement units.
	Pressure	Provides access to the unit converter for pressure measurement units.
	Speed	Provides access to the unit converter for speed measurement units.
	Temperature	Provides access to the unit converter for temperature measurement units.
	Time	Provides access to the unit converter for time measurement units.
	Volume	Provides access to the unit converter for volume measurement units.

Public Methods

	Name	Description
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)

	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[Units Members](#)
[DevExpress.UnitConversion Namespace](#)

Area Property

Provides access to the unit converter for area measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Area As AreaUnitsConverter
```

C#

```
public static AreaUnitsConverter Area { get; }
```

Property Value

A [AreaUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Binary Property

Provides access to unit converter for binary metric prefixes.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Binary As BinaryUnitsConverter
```

C#

```
public static BinaryUnitsConverter Binary { get; }
```

Property Value

A [BinaryUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Distance Property

Provides access to the unit converter for distance measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Distance As DistanceUnitsConverter
```

C#

```
public static DistanceUnitsConverter Distance { get; }
```

Property Value

A [DistanceUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Energy Property

Provides access to the unit converter for energy measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Energy As EnergyUnitsConverter
```

C#

```
public static EnergyUnitsConverter Energy { get; }
```

Property Value

A [EnergyUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Force Property

Provides access to the unit converter for force measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Force As ForceUnitsConverter
```

C#

```
public static ForceUnitsConverter Force { get; }
```

Property Value

A [ForceUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Information Property

Provides access to the unit converter for information measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Information As InformationUnitsConverter
```

C#

```
public static InformationUnitsConverter Information { get; }
```

Property Value

A [InformationUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Magnetism Property

Provides access to the unit converter for magnetic field measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Magnetism As MagnetismUnitsConverter
```

C#

```
public static MagnetismUnitsConverter Magnetism { get; }
```

Property Value

A [MagnetismUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Mass Property

Provides access to the unit converter for mass measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Mass As MassUnitsConverter
```

C#

```
public static MassUnitsConverter Mass { get; }
```

Property Value

A [MassUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Metric Property

Provides access to the unit converter for metric prefixes.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Metric As MetricUnitsConverter
```

C#

```
public static MetricUnitsConverter Metric { get; }
```

Property Value

A [MetricUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Power Property

Provides access to the unit converter for power measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Power As PowerUnitsConverter
```

C#

```
public static PowerUnitsConverter Power { get; }
```

Property Value

A [PowerUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Pressure Property

Provides access to the unit converter for pressure measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Pressure As PressureUnitsConverter
```

C#

```
public static PressureUnitsConverter Pressure { get; }
```

Property Value

A [PressureUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Speed Property

Provides access to the unit converter for speed measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Speed As SpeedUnitsConverter
```

C#

```
public static SpeedUnitsConverter Speed { get; }
```

Property Value

A [SpeedUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Temperature Property

Provides access to the unit converter for temperature measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Temperature As TemperatureUnitsConverter
```

C#

```
public static TemperatureUnitsConverter Temperature { get; }
```

Property Value

A [TemperatureUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Time Property

Provides access to the unit converter for time measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Time As TimeUnitsConverter
```

C#

```
public static TimeUnitsConverter Time { get; }
```

Property Value

A [TimeUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Volume Property

Provides access to the unit converter for volume measurement units.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
static Public ReadOnly Property Volume As VolumeUnitsConverter
```

C#

```
public static VolumeUnitsConverter Volume { get; }
```

Property Value

A [VolumeUnitsConverter](#) object.

See Also

[Units Class](#)

[Units Members](#)

[DevExpress.UnitConversion Namespace](#)

Volume Enumeration

Lists units of volume measurement.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax
Visual Basic
`Public Enum Volume`
C#
`public enum Volume`

Members

Name	Description
Bbl	U.S. oil barrel ("barrel").
Bushel	U.S. bushel ("bushel").
CubicAngstrom	Cubic angstrom ("ang3").
CubicFoot	Cubic feet ("ft3").
CubicInch	Cubic inch ("in3").
CubicLightYear	Cubic light-year ("ly3").
CubicMeter	Cubic meter ("m3").
CubicMile	Cubic Mile ("mi3").
CubicMileNautical	Cubic nautical mile ("Nmi3").
CubicPicaPoint	Cubic Pica ("Pica3").
CubicYard	Cubic yard ("yd3").
Cup	Cup ("cup").
Gallon	Gallon ("gal").
GallonImperial	Imperial gallon (U.K.) ("uk_gal").
GrossRegisteredTon	Gross Registered Ton ("GRT").
GRT	Gross Registered Ton ("GRT").
Liter	Liter ("lt").
MeasurementTon	Measurement ton (freight ton) ("MTON").
Mton	Measurement ton (freight ton) ("MTON").
OilBarrel	U.S. oil barrel ("barrel").
OunceFluid	Fluid ounce ("oz").
Oz	Fluid ounce ("oz").
Pint	U.S. pint ("pt").
PintImperial	U.K. pint ("uk_pt").
Quart	Quart ("qt").

QuartImperial	Imperial quart (U.K.) ("uk_qt").
Regton	Gross Registered Ton ("GRT").
Tablespoon	Tablespoon ("tbs").
Teaspoon	Teaspoon ("tsp").
TeaspoonModern	Modern teaspoon ("tspm").

See Also
[DevExpress.UnitConversion Namespace](#)

VolumeUnitsConverter Class

Converts volume measurement from one unit to another.

Namespace: [DevExpress.UnitConversion](#)
Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

Public Class VolumeUnitsConverter Inherits [BaseUnitsConverter\(of Volume\)](#)

C#

public class VolumeUnitsConverter : [BaseUnitsConverter<Volume>](#)

Remarks

Use the [PrefixUnitsConverter<T>.Convert](#) method to perform the conversion.

To convert 100 barrels to liters, use the following code:

C#	
<pre>double result = Units.Volume.Convert(100.0f, Volume.OilBarrel, Volume.Liter);</pre>	

Visual Basic

<pre>Dim result As Double = Units.Volume.Convert(100.0f, Volume.OilBarrel, Volume.Liter)</pre>	

Inheritance Hierarchy

[Object](#)
 [BaseUnitsConverter<Volume>](#)
 VolumeUnitsConverter

See Also

[VolumeUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Volume](#)

VolumeUnitsConverter Members








Converts volume measurement from one unit to another.

The following tables list the members exposed by the [VolumeUnitsConverter](#) type.

Public Constructors

	Name	Description
	VolumeUnitsConverter	Initializes a new instance of the VolumeUnitsConverter class with the default settings.

Public Methods

	Name	Description
	Convert	Overloaded. Converts a value of type Double from one unit of measurement to another. (Inherited from BaseUnitsConverter<T>)
	Equals	Determines whether the specified System.Object is equal to the current System.Object . (Inherited from Object)
	Equals	Determines whether the specified System.Object instances are considered equal. (Inherited from Object)
	GetHashCode	Serves as a hash function for a particular type. System.Object.GetHashCode is suitable for use in hashing algorithms and data structures like a hash table. (Inherited from Object)
	GetTransform	Obtains a delegate function that performs the conversion. (Inherited from BaseUnitsConverter<T>)
	GetType	Gets the System.Type of the current instance. (Inherited from Object)
	ReferenceEquals	Determines whether the specified System.Object instances are the same instance. (Inherited from Object)
	ToString	Returns a System.String that represents the current System.Object . (Inherited from Object)

See Also
[VolumeUnitsConverter Members](#)
[DevExpress.UnitConversion Namespace](#)
[Volume](#)

VolumeUnitsConverter Constructor

Initializes a new instance of the [VolumeUnitsConverter](#) class with the default settings.

Namespace: [DevExpress.UnitConversion](#)

Assembly: DevExpress.Docs.v18.1.dll

Syntax

Visual Basic

```
Public New()
```

C#

```
public VolumeUnitsConverter()
```

See Also

[VolumeUnitsConverter Class](#)

[VolumeUnitsConverter Members](#)

[DevExpress.UnitConversion Namespace](#)